MARIN VLADA
IOAN NISTOR
ADRIAN POSEA
CĂLIN CONSTANTINESCU

GRAFICĂ PE CALCULATOR ÎN LIMBAJELE PASCAL ȘI C

Implementare și aplicații

volumul II: APLICAȚII



EDITURA TEHNICĂ,

MARIN VLADA
IOAN NISTOR

ADRIAN POSEA
CĂLIN CONSTANTINESCU

GRAFICĂ PE CALCULATOR ÎN LIMBAJELE PASCAL ȘI C

Implementare și aplicații

volumul II: APLICAȚII





EDITURA TEHNICĂ,
BUCUREȘTI 1992

Adresa: **Editura Tehnică**
Piaţa Presei Libere, 1
33 Bucureşti, România
cod 79738

Control ştiinţific:
conf. dr. **HORIA GEORGESCU**,
conf. dr. **OCTAVIAN BÂSCĂ**

Redactor:
ing. **SILVIA CÂNDEA**

Coperta:
ADRIAN AVRAM

Tehnoredactare computerizată:
mat. **GHEORGHE SĂVULESCU**

Bun de tipar: 25.09.1991
Coli de tipar: 16,25
C.Z.: 76:681,3
Comanda nr. 411/1991
IMPRIMERIA "ARDEALUL" CLUJ



ISBN 973-31-0406-X
ISBN 973-31-0408-6

CUPRINS GENERAL

Volumul I

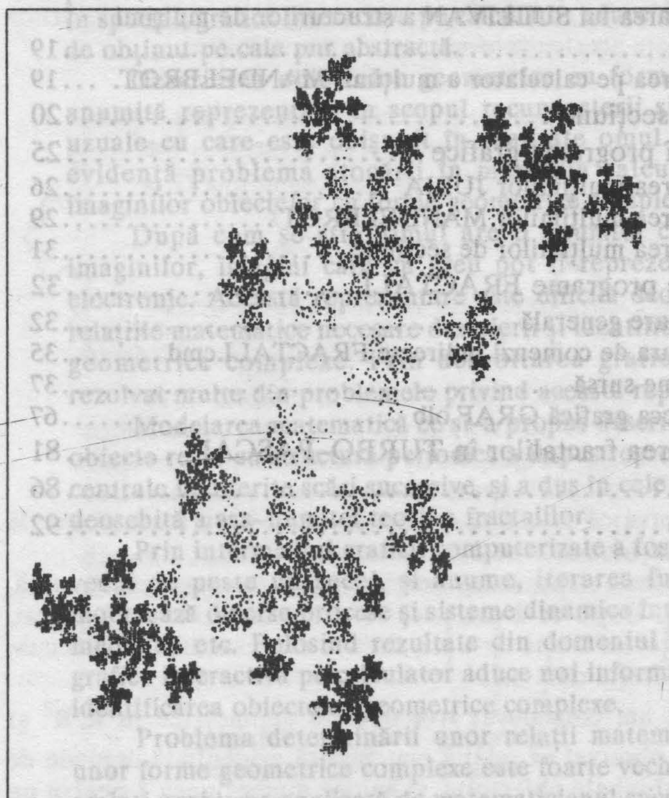
Volumul II

CUPRINS

Volumul I

APLICAȚII ÎN TEORIA FRACTALILOR

Haos și fractali



5

5. APLICAŢII ÎN TEORIA FRACTALILOR

5

5.1. Introducere

În contextul revoluției actuale a informaticii, calculatorul se recomandă ca un instrument valoros în activitatea de cercetare și în particular în cercetarea matematică cu aplicabilitate în știință și tehnică. În special, grafica interactivă pe calculator aduce noi informații mai dificil de obținut pe cale pur abstractă.

Descrierea obiectelor geometrice cu forme complexe necesită o anumită reprezentare în scopul recunoașterii și identificării formelor uzuale cu care este obișnuit în realitate omul. Acest lucru a scos în evidență problema stocării în memoria calculatorului electronic a imaginilor obiectelor cu forme geometrice complexe.

După cum se știe, omul are o anumită capacitate de analiză a imaginilor, imagini care cu greu pot fi reprezentate pe un calculator electronic. Această reprezentare este dificilă deoarece trebuie precizate relațiile matematice necesare descrierii și identificării obiectelor cu forme geometrice complexe. Prin dezvoltarea graficii computerizate s–au rezolvat multe din problemele privind această reprezentare.

Modelarea matematică ce și–a propus descrierea unor fenomene sau obiecte reale cu structură periodică a impus reprezentarea unor elemente centrale la diferite scări succesive, și a dus în cele din urmă la dezvoltarea deosebită a așa–numitei teorii a fractalilor.

Prin intermediul graficii computerizate a fost revitalizat un domeniu vechi de peste un secol, și anume, iterarea funcțiilor complexe care modelează diverse procese și sisteme dinamice întîlnite în fizică, biologie, medicină etc. Folosind rezultate din domeniul științei calculatoarelor, grafica interactivă pe calculator aduce noi informații privind descrierea și identificarea obiectelor geometrice complexe.

Problema determinării unor relații matematice pentru descrierea unor forme geometrice complexe este foarte veche. Ca exemplu, se poate aminti problema analizată de matematicianul suedez HERGE von KOCH care în anul 1904 a enunțat următoarea construcție geometrică (forma geometrică cunoscută sub numele de „curba lui KOCH" cu proprietatea

că are o lungime infinită şi delimitează o suprafaţă de arie finită):

- se pleacă de la un triunghi echilateral cu latura avînd o lungime finită;
- fiecare latură se împarte în 3 părţi egale, segmentul din mijloc se elimină şi se înlocuieşte cu un unghi ale cărui laturi sînt formate din două segmente de lungime egală cu segmentul eliminat;
- pentru fiecare segment al figurii obţinute se repetă operaţia anterioară;
- dacă se repetă acest proces, se obţine o linie poligonală închisă cu latura de lungime din ce în ce mai mică.

Se poate observa că după fiecare iteraţie lungimea curbei creşte, la limită aceasta tinde la infinit, dar suprafaţa delimitată de această linie poligonală are arie finită (vezi fig. 5–1).

Construcţia este perfect regulată şi după fiecare trecere de la un nivel la altul, numărul de unghiuri creşte, de asemenea numărul de laturi ale liniei poligonale şi lungimea totală a curbei cresc, dar lungimile laturilor se micşorează şi toate acestea într–un raport constant.

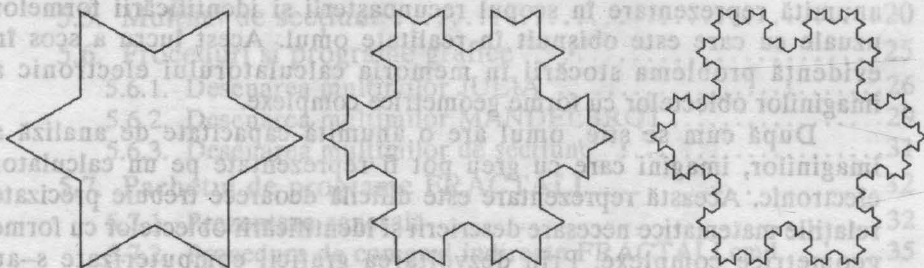


Figura 5–1.

Se poate observa că un astfel de raport există între toate obiectele geometrice de aceeaşi formă cînd dimensiunile lor cresc.

De exemplu, dacă se dublează lungimea unui segment de dreaptă, atunci lungimea lui creşte de 2 ori, dacă se dublează latura unui pătrat, atunci aria suprafeţei lui creşte de 4 ori, dacă se dublează latura unui cub, atunci volumul lui creşte de 8 ori.

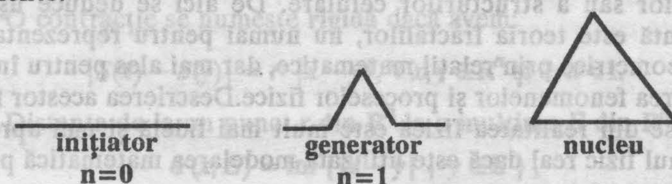
Prin urmare, am considerat trei obiecte geometrice în $\mathbf{R}, \mathbf{R}^2$ şi respectiv $\mathbf{R}^3$ şi am observat că relaţia matematică evidenţiată depinde de numărul de dimensiuni ale obiectului, şi anume dimensiunea 1 pentru un segment de dreaptă din $\mathbf{R}$, dimensiunea 2 pentru o suprafaţă din $\mathbf{R}^2$ şi dimensiunea 3 pentru un corp din $\mathbf{R}^3$.

Pentru curba lui KOCH să notăm cu $l(n)$, $n=0,1,2,\ldots$ lungimea curbei la iterația n. Dacă $p = l(0)$ reprezintă perimetrul triunghiului echilateral și făcînd observația că în procesul de construcție fiecare latură se împarte în 3 părți egale și se transformă într-o linie formată din 4 segmente, atunci, evident, avem:

$$l(n) = p\left(\frac{4}{3}\right)^n, n = 0,1,2,\ldots$$

$$\lim_{n\to\infty} l(n) = \infty$$

Pentru realizarea construcției geometrice, se disting următoarele elemente:

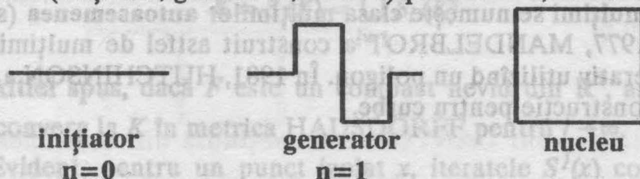


- s = factorul de scară,
- t = factorul de descompunere
- D = dimensiunea fractală, unde

$$(1/s)^D = t \Leftrightarrow D = \frac{\lg t}{\lg(1/s)}$$

Pentru cazul particular prezentat, avem $s = 1/3$, $t = 4$ și dimensiunea fractală $D = \frac{\lg 4}{\lg 3} = 1{,}2619$

Prin urmare, în cazul curbei KOCH prezentat mai sus, lungimea curbei se mărește după fiecare iterație de $4/3$ ori, avînd o lege proprie de creștere, determinată de principiul de construcție. Forma geometrică obținută este o construcție geometrică cuprinsă între o curbă și o suprafață căreia i se poate asocia numărul fracționar $\lg 4/\lg 3$ ce reprezintă dimensiunea fractală. Pentru alte construcții geometrice, cele trei elemente (inițiator, generator, nucleu) pot fi altfel, de exemplu:



unde:

$$s = \frac{1}{4}, \quad t = 8, \quad D = \frac{\lg 8}{\lg 4} = 1{,}5$$

şi astfel se va obţine o altă construcţie geometrică.

Noţiunea de dimensiune fractală a fost introdusă în anul 1919 de matematicianul HAUSDORFF. Pornind de la această modelare matematică s–au putut reprezenta matematic forme geometrice deosebit de complexe. Deoarece, în reprezentarea fractală forma obiectului se obţine printr–o repetare a unui nucleu central, memoria utilizată de calculator pentru stocarea imaginilor respective este deosebit de redusă, în schimb sînt disponibile relaţiile matematice cu care se poate recunoaşte un obiect sub diferite forme. De exemplu, spaţiul de memorie necesar pentru unele reprezentări poate fi de 100, 1000,... ori mai mic.

Viaţa însăşi, sub diferitele ei forme este generată pe baza informaţiilor genetice conţinute în nucleu prin repetarea la diferite scări a celulelor sau a structurilor celulare. De aici se deduce uşor cît de importantă este teoria fractalilor, nu numai pentru reprezentarea unor forme geometrice prin relaţii matematice, dar mai ales pentru înţelegerea şi studierea fenomenelor şi proceselor fizice.Descrierea acestor fenomene şi procese din realitatea fizică este mult mai fidelă şi mai apropiată de fenomenul fizic real dacă este utilizată modelarea matematică prin teoria fractalilor.

Studii şi rezultate importante s–au obţinut în domeniul sistemelor dinamice prin contribuţii de excepţie referitoare la procese de forma:

$$Z_{n+1} = f(Z_n), Z_0 \text{ dat şi}$$

$$Z_n \in \mathbb{C} \text{ (planul complex)}$$

Contribuţii importante în acest domeniu au fost aduse de GASTON JULIA (1893 - 1978) şi PIERRE FATOU (1878 - 1929) , BENOIT B. MANDELBROT, DENNIS SULLIVAN, ANDRIEN DOUAY, ROBERT L. DEVANEY şi JOHN HAMAL HUBBARD, HERMANN WEYL şi CARL LUDWIG SIEGEL.

5.2. Aproximaţii de fractali

O clasă specială de fractali este dată de fractalii obţinuţi prin construcţie matematică şi prin utilizarea graficii computerizate. Această clasă de mulţimi se numeşte clasa mulţimilor **autoasemenea** (similare).

În 1977, MANDELBROT a construit astfel de mulţimi pentru un proces iterativ utilizînd un poligon. În 1981, HUTCHINSON a generalizat această construcţie pentru curbe.

5.2.1. Aproximarea mulțimilor compacte

În $\mathbf{R}^n$ se consideră o contracție S, adică

$$| S(x) - S(y) | \leq c \, | x - y |, \; \forall x, y \in \mathbf{R}^n, \; 0 < c < 1.$$

Valoarea minimă pentru c care verifică inegalitatea amintită se numește **rația contracției**. O mulțime compactă $E \subset \mathbf{R}^n$ este invariantă pentru o mulțime finită de contracții din $\mathbf{R}^n$, $S^* = \{ S_1, S_2, \ldots, S_m \}$ dacă avem:

$$E = \bigcup_{i=1}^{m} S_i (E)$$

O contracție se numește **rigidă** dacă avem:

$$|S(x) - S(y)| = r \cdot |x - y|, \; \forall x, y \in \mathbf{R}^n \text{ și } 0 < r < 1.$$

Distanța de la un punct x din $\mathbf{R}^n$ la o mulțime E din $\mathbf{R}^n$ este dată de

$$d(x, E) = \inf \{ |x - y| \mid y \in E \},$$

iar metrica HAUSDORFF δ **între două mulțimi nevide din $\mathbf{R}^n$ este dată de**

$$\delta(E, F) = \sup \{ d(x, F), d(y, E) \mid x \in E, y \in F \}.$$

Spațiul tuturor **mulțimilor compacte din $\mathbf{R}^n$ este un spațiu metric H cu metrica δ. Fie aplicația:**

$$S : E \to \bigcup_{i=1}^{m} S_i (E)$$

ce reprezintă **o contracție a lui H.** Punctul fix al lui S, notat K este o mulțime **invariantă pentru mulțimea** de contracții $\{S_i | i=\overline{1,m}\}$. Familia $\{S_i | i=\overline{1,m}\}$ **se numește familia generatoare pentru mulțimea** K.

Teoremă: ***Fie*** $S^* = \{S_1, S_2, \ldots, S_m\}$ *o familie finită de contracții din* $\mathbf{R}^n$; *atunci* ***există o unică mulțime*** *compactă K astfel că avem:*

$$K = S(K) = \bigcup_{i=1}^{m} S_i (K)$$

Altfel spus, dacă F este un compact nevid din $\mathbf{R}^n$, atunci **iteratele** $S^j(F)$ **converg** la K în metrica HAUSDORFF pentru $j \to \infty$.

Evident, pentru un punct izolat x, iteratele $S^j(x)$ converg la K în metrica HAUSDORFF și $S^j(x)$ are cel mult m^j elemente. În practică, trebuie găsită familia de contracții care generează pe K și trebuie să se reprezinte grafic K prin plotarea iteratelor $S^j(x)$ de la un moment dat (de

exemplu, de la iterata 1000). Problema complicată este cum să se determine contracţiile care generează pe K.

Dacă mulţimea K este invariantă pentru familia de contracţii $\{S_i | i=\overline{1,m}\}$, iar $\{r_i | i=\overline{1,m}\}$ reprezintă mulţimea raţiilor acestor contracţii, atunci dimensiunea de autoasemănare (similitudine) a lui K este unicul număr pozitiv s astfel ca

$$\sum_{i=1}^{m} r_i^s = 1$$

Dacă $H^s\left(S_i(E) \cap S_j(E)\right) = 0$, $i \neq j$, atunci K se numeşte mulţime **autoasemenea**, unde H^s este s–măsura HAUSDORFF. Autoasemănarea reprezintă compunerea dintre o translaţie şi o omotetie.

Dacă $E \subset \mathbb{R}^n$ şi $E \subset \bigcup_{i=1}^{\infty} U_i$, unde $0 < |U_i| =< \delta$, $\forall\, i=1, 2, 3, \dots$ vom spune că familia $\{U_i | i=1,2,\dots\}$ este δ–acoperirea lui E, unde $\delta>0$.

Pentru $\delta>0$, se defineşte

$$H^s_\delta(E) = \inf \sum_{i=1}^{\infty} |U_i|^s,$$

unde infimumul se consideră pentru orice δ–acoperire $\{U_i | i = 1,2,\dots\}$ a lui E. Am notat prin $|U|$ diametrul lui U, adică

$$|U| = \sup\{\, |x-y| \mid x,y \in U\}.$$

Se defineşte s–măsura HAUSDORFF a lui E, numărul

$$H(E) = \lim_{\delta \to 0} H^s(E)$$

Dimensiunea HAUSDORFF a lui E se defineşte prin numărul

$$\dim(E) = \inf\{s \mid H^s(E) = 0\}.$$

Teoremă: *Dacă K este mulţime autoasemenea, atunci dimensiunea HAUSDORFF a lui K este identică cu dimensiunea de autoasemănare a lui K.*

Exemplul 1:

Mulţimea lui CANTOR este obţinută din mulţimea compactă invariantă pentru contracţiile $\{S_1, S_2\}$, unde $S_1(x) = \frac{x}{3}$, $S_2(x) = \frac{(x+2)}{3}$. Evident, raţiile acestor contracţii sînt $r_1 = r_2 = 1/3$ şi astfel dimensiunea de asemănare s verifică relaţia $2(1/3)^s = 1$, adică $s = \lg 2 / \lg 3$.

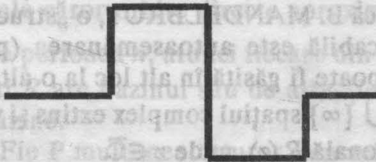
Exemplul 2:

Clasica curbă KOCH este generată de patru omotetii care au rațiile egale cu ⅓. Conform teoremei, dimensiunea HAUSDORFF este egală cu $s = \frac{\lg 4}{\lg 3} = 1{,}2619$, deoarece $r_1 = r_2 = r_3 = r_4 = \frac{1}{3}$, iar generatorul are forma:



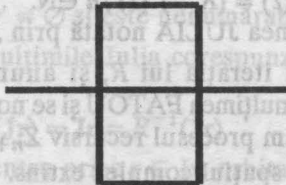
Exemplul 3:

Curba lui KOCH corespunzătoare unui pătrat (nucleul este un pătrat) este generată de opt omotetii ce au rațiile contracțiilor egale cu ¼. Prin urmare, dimensiunea HAUSDORFF este egală cu $s = \frac{\lg 8}{\lg 3} = 1{,}5$, deoarece $r_1 = r_2 = \ldots = r_8 = \frac{1}{4}$, iar generatorul are forma:



Exemplul 4:

Curba PEANO este generată de nouă omotetii ce au rațiile contracțiilor egale cu ⅓. Prin urmare, dimensiunea HUSDORFF este egală cu $s = \frac{\lg 9}{\lg 3} = 2$, deoarece $r_1 = r_2 = \ldots = r_9 = \frac{1}{3}$, iar generatorul are forma:



5.3. Mulțimile JULIA și proprietățile lor

Să considerăm în planul complex următorul proces recursiv:

$$Z_{n+1} = f(Z_n) = Z_n^2 + c, \text{ cu } f: \mathbb{C} \to \mathbb{C},\ Z_0, c \in \mathbb{C},\ n \in \mathbb{N}.$$

În ciuda formei sale algebrice foarte simple, acest proces prezintă o dinamică extrem de sofisticată, uneori haotică.

Dacă, pentru început, $c=0$, sînt posibile trei situații pentru șirul (Z_n):

- dacă $|Z_0| < 1$, atunci $|Z_n| \to 0$ și spunem că 0 se numește atractor al procesului.
- dacă $|Z_0| > 1$, atunci $|Z_n| \to \infty$ și spunem că ∞ se numește atractor al procesului.
- dacă $|Z_0| = 1$, atunci $|Z_n| = 1$, pentru orice $n \in \mathbb{N}$; șirul are deci termenii situați pe frontiera dintre două domenii de atracție, în cazul nostru cercul unitate.

Cercul unitate este cel mai simplu exemplu de mulțime JULIA. Dacă modificăm c, luînd de exemplu $c = -0{,}12375 + 0{,}56508i$, 0 nu mai rămîne atractor interior, iar frontiera dintre domeniile de atracție devine foarte neregulată, după cum specifică B. MANDELBROT, o „structură fractală".

Proprietatea ei remarcabilă este autoasemănarea (proprietate de hologramă). Aceeași formă poate fi găsită în alt loc la o altă mărime.

Vom nota prin $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ spațiul complex extins și vom considera în acest spațiu o funcție rațională $R(z)$, unde $z \in \overline{\mathbb{C}}$.

Fie $R(z) = \dfrac{P(z)}{Q(z)}$ o funcție rațională în planul complex extins, unde $P, Q \in \overline{\mathbb{C}}[X]$, P, Q fiind prime între ele, iar gradul lui R este $d = \text{grad}(R) = \max\{\text{grad}(P), \text{grad}(Q)\}$.

Preimaginea lui z prin R, o vom nota prin $R^{-1}(z) = \{v \in \overline{\mathbb{C}} \mid R(v) = z\}$.

Vom nota prin $R^n(z) = (R \circ R \circ \ldots \circ R)(z)$, $n \in \mathbb{N}$, compunerea făcîndu-se de n ori, iar prin $R^{-n}(z) = (R^{-1})^n(z)$, $n \in \mathbb{N}$.

Într-un anumit sens, mulțimea JULIA notată prin **Jr** este mulțimea punctelor „excepționale" pentru iterația lui R, și anume pentru $R^n(z)$. Complementara lui **Jr** se numește mulțimea FATOU și se notează **Fr** $= \overline{\mathbb{C}} \setminus$ **Jr**.

Pentru $Z_0 \in \overline{\mathbb{C}}$ dat, considerăm procesul recursiv $Z_{n+1} = R(Z_n)$, $n \in \mathbb{N}$ ce definește un șir de puncte din spațiul complex extins.

DEFINIȚII:

a) Pentru $Z_0 \in \mathbb{C}$, $Or^+(Z_0) = \{Z_n \mid n \in \mathbb{N}\}$ se numește **orbită directă** a lui Z_0.

b) Pentru $Z_0 \in C$, $Or^-(Z_0) = \{ R^{-k}(Z_0) \mid k \in N \}$ se numeşte **orbită inversă** a lui Z_0.

c) Pentru $Or^+(Z_0)$ finită, adică dacă există n ∈ N, cu $Z_n = Z_0$, orbita se numeşte ciclu sau orbită periodică de perioadă n, iar punctul Z_0 punct periodic.

d) Dacă $Or^+(Z_0)$ este orbită periodică de perioada n, atunci pentru orice $Z \in Or^+(Z_0)$, $\lambda = R'(Z)$ este o constantă, numită valoare proprie a lui Z_0.

e) Un punct periodic se numeşte:

atractiv	⇔	$0 < \lvert\lambda\rvert < 1$
indiferent	⇔	$\lvert\lambda\rvert = 1$
superatractiv	⇔	$\lambda = 0$
repulsiv	⇔	$\lvert\lambda\rvert > 1$

f) Dacă Z_0 este un punct fix atractiv al lui R, deci $R(Z_0) = Z_0$, se numeşte **bazinul de atracţie** al lui Z_0, mulţimea $A(Z_0) = \{z \in \overline{C} \mid R^k(Z_0) \to Z_0 \text{ cînd } k \to \infty\}$. $A(Z_0)$ colectează toate punctele ale căror orbite directe aproximează pe Z_0. Dacă γ este un ciclu atractiv de perioadă n, atunci fiecare din punctele fixe $R^i(Z_0)$, $i=0, 1, 2, ..., n-1$ ale lui R are bazinul său de atracţie, iar $A(\gamma)$ este de fapt reuniunea acestor bazine.

g) Fie **P** mulţimea tuturor punctelor repulsive. G. JULIA a definit mulţimea **Jr** (mulţimea JULIA) ca fiind **Jr=P'**, unde **P'** este mulţimea derivată a lui **P**. Într-un anumit sens, mulţimea JULIA este mulţimea punctelor excepţionale ale iteratelor lui R.

5.3.1. Proprietăţi fundamentale ale mulţimii JULIA

a). **P** este densă în **Jr**.

b). **Jr'** $\neq \emptyset$ şi este nenumărabilă.

c). Mulţimile Julia corespunzătoare lui R şi lui R^k, $k = 1,2...$, sînt identice.

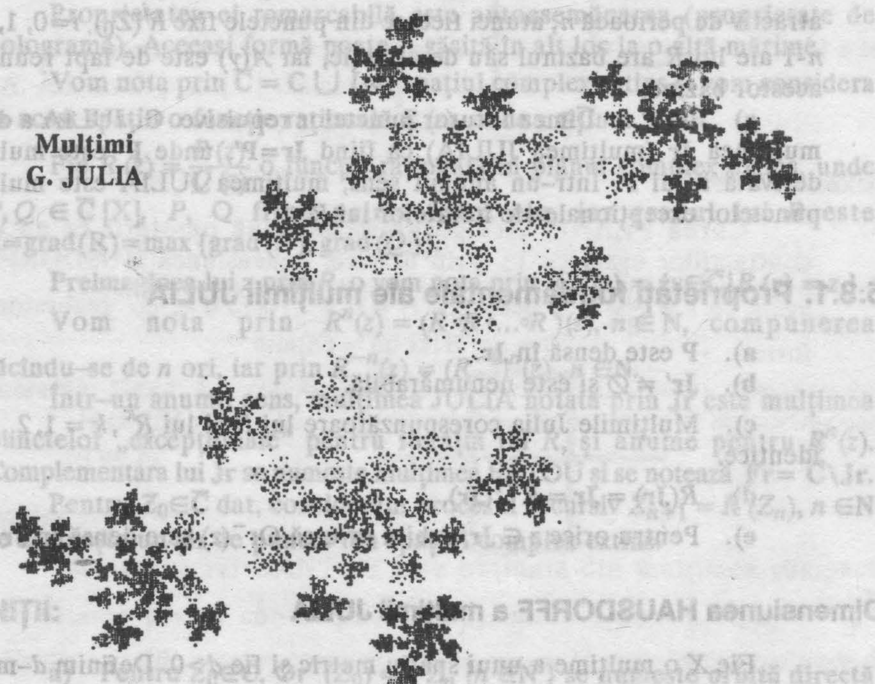
d). $R(\text{Jr}) = \text{Jr} = R^{-1}(\text{Jr})$.

e). Pentru orice $z \in$ **Jr**, orbita inversă $Or^-(z)$ este densă în **Jr**.

Dimensiunea HAUSDORFF a mulţimii JULIA

Fie X o mulţime a unui spaţiu metric şi fie $d>0$. Definim d-măsura exterioară a mulţimii X ca fiind:





Mulţimi
G. JULIA

$$m_d(X) = \liminf_{\varepsilon \to 0} \left\{ \sum_{i \in I} (\text{diam} S_i)^d \mid X \subseteq \bigcup_{i \in I} S_i,\ I \text{ finită}, \text{ unde } S_i \text{ sînt bile cu diam } S_i < \varepsilon \right\}$$

$m_d(X)$ depinde de alegerea lui d, putînd fi finită sau infinită.

În 1919, matematicianul HAUSDORFF demonstrează că există un unic $d = d^*$ astfel încît $m_d(X)$ trece de la valoarea infinită la valoare finită, cînd d creşte. Dimensiunea Hausdorff a mulţimii X se defineşte astfel:

$$h(X) = \sup \{ d \in R_+ \mid m_d(X) = \infty \}$$

Intuitiv, $h(X)$ măsoară mărimea numărului de mulţimi de diametre mai mici decît ε necesare pentru a acoperi pe X, cînd $\varepsilon \to 0$. O mulţime se numeşte **fractal**, dacă dimensiunea sa Hausdorff $h(X)$ nu este număr întreg.

Mulţimile JULIA sînt fractali. Mai mult, pentru $|c|<1$ şi $f(z) = z^2 + c$, $h(J_c) = 1 + \frac{|c|}{4 \log 2}$, unde J_c este mulţimea JULIA corespunzătoare funcţiei $f(z)$.

Pentru c mic, J_c este o curbă Jordan. Deşi mulţimile JULIA sînt de natură tipic fractală, aproape nimic nu este cunoscut despre dimensiunea lor HAUSDORFF.

O dezvoltare a acestei problematici se află în [2].

5.3.2. Metoda BSM (Boundary Scanning Method) pentru reprezentarea grafică a mulţimilor JULIA

Metoda BSM reprezintă metoda analizei frontierelor în scopul reprezentării cu ajutorul calculatorului electronic a mulţimilor JULIA. Dacă a este un punct fix atractiv, $|R'(a)|<1$, $R(a) = a$, iar $A(a)$ este bazinul său de atracţie, atunci $Jr = \partial A(a)$. Fie o mulţime normă $L_0(a)$ astfel încît $a \in L_0(a) \subseteq A(a)$.

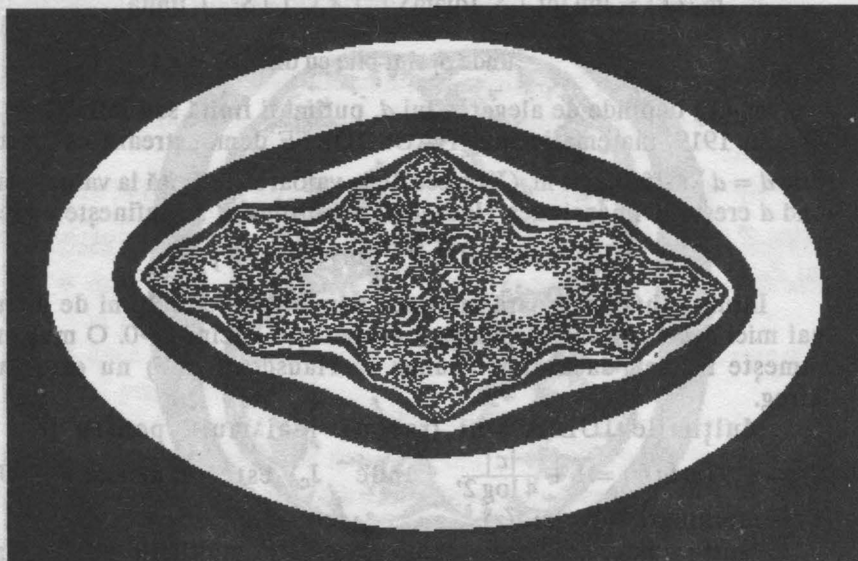
Definim mulţimile de nivel de atracţie egală în $A(a)$ în raport cu mulţimea normă $L_0(a)$, ca fiind

$$L_k(a) = \{ z \mid R^k(z) \in L_0(a) \text{ şi } R^2(z) \notin L_0(a) \text{ pentru } l < k \},\ k = 1,2,...$$

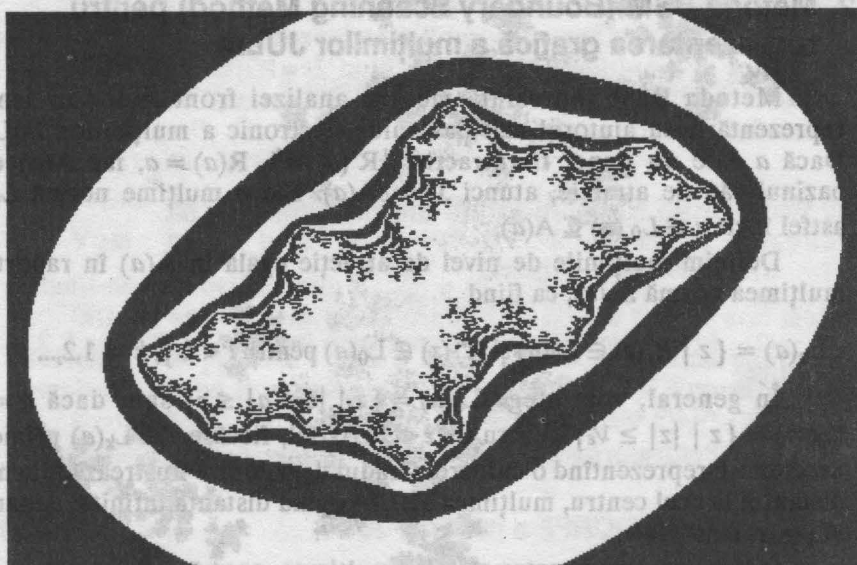
În general, vom alege $L_0(a) = \{ z \mid |z - a| \leq \varepsilon \}$ sau, dacă $a = \infty$, $L_0(\infty) = \{ z \mid |z| \geq 1/\varepsilon \}$ pentru $0<\varepsilon \ll 1$. Astfel, fiecare $z \in L_k(a)$ primeşte un indice k reprezentînd o culoare. Gradul de colorare ilustrează dinamica distanţei la acel centru, mulţimea JULIA avînd distanţa infinită, deoarece $\partial L_k \to Jr$, cînd $k \to \infty$.

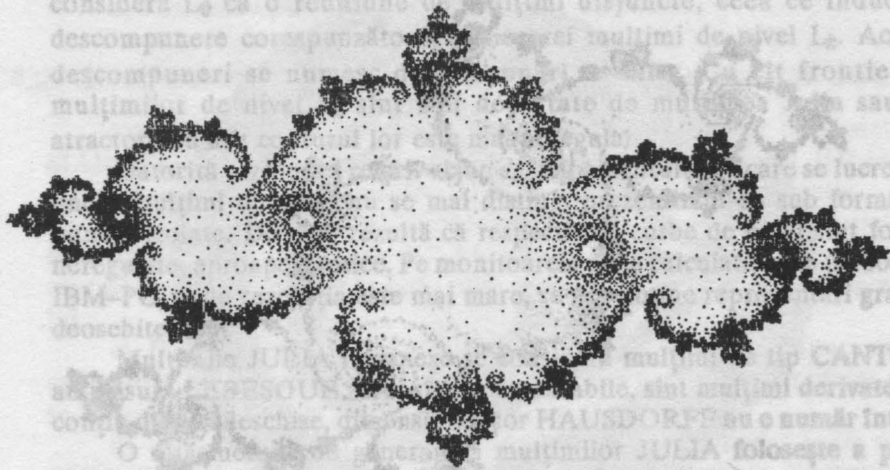
În experienţele noastre am ales mulţimea–normă

$$L_0(\infty) = \{ c \mid |c| > 1/\varepsilon \} \text{ pentru } 0<\varepsilon \ll 1.$$

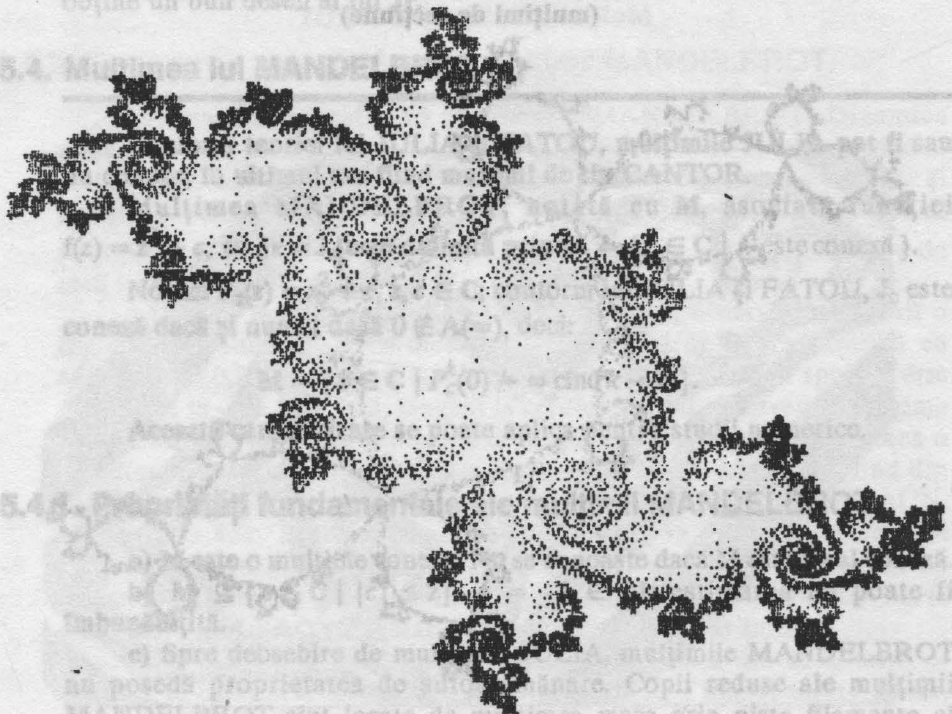


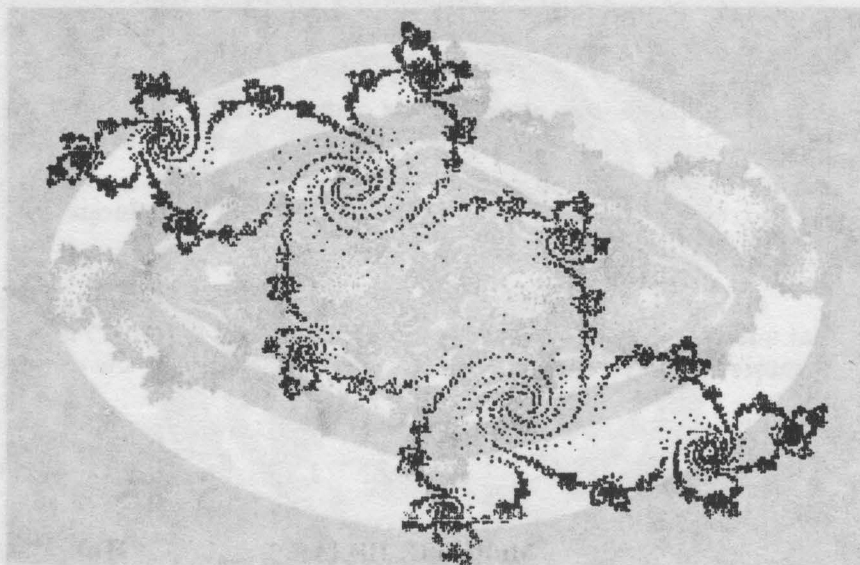
Mulţimi G. JULIA



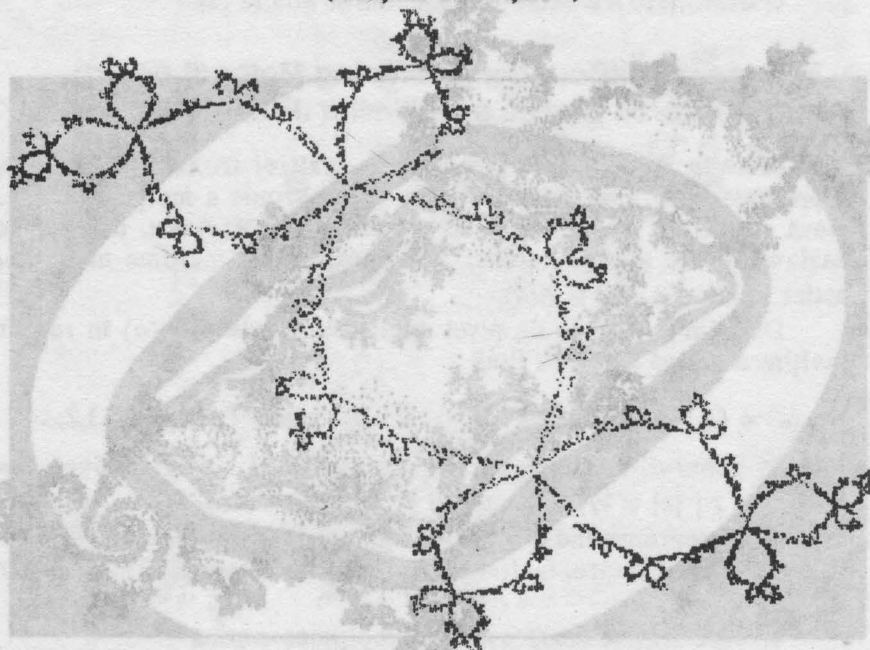


Mulțimi G. JULIA
(mulțimi de secțiune)





Mulțimi G. JULIA
(mulțimi de secțiune)



Mulţimea–normă L_0 poate fi aleasă arbitrar. În particular, putem considera L_0 ca o reuniune de mulţimi disjuncte, ceea ce induce o descompunere corespunzătoare a fiecărei mulţimi de nivel L_k. Aceste descompuneri se numesc **descompuneri** *m*–**adice**. Cu cît frontierele mulţimilor de nivel L_k sînt mai depărtate de mulţimea Julia sau de atractori, cu atît conturul lor este mai neregulat.

Datorită rezoluţiei relativ slabe a echipamentului cu care se lucrează, unele mulţimi de nivel nu se mai disting, prezentîndu–se sub formă de puncte izolate. De aici, rezultă că respectivele curbe de nivel sînt foarte neregulate, aproape haotice. Pe monitoarele de la calculatoarele personale IBM–PC, unde rezoluţia este mai mare, se pot obţine reprezentări grafice deosebite.

Mulţimile JULIA neconexe se consideră mulţimi **de tip CANTOR:** au măsura LEBESGUE zero, sînt nenumărabile, **sînt mulţimi derivate, nu** conţin discuri deschise, dimensiunea lor HAUSDORFF **nu e număr întreg.**

O altă metodă de generare a mulţimilor JULIA **foloseşte a patra** proprietate fundamentală a mulţimilor Julia. **Reprezentînd grafic** mulţimile $Jr^n = \{z \in C \mid \exists k \le n \text{ cu } R^n(z) = \bar{z}\}$, **unde** $\bar{z} \in Jr$ **este un punct** periodic repulsiv fixat, **deducem că pentru un** *n* **suficient de mare vom** obţine un bun desen al lui **Jr.**

5.4. Mulţimea lui MANDELBROT

Conform teoriei lui JULIA şi FATOU, mulţimile JULIA pot fi sau nu conexe, **în ultimul caz fiind mulţimi de tip CANTOR.**

Mulţimea MANDELBROT, notată cu M, asociată funcţiei $f(z) = z^2 + c$, **cu** $Jr = J_c$ **este definită prin** $M = \{c \in C \mid J_c \text{ este conexă}\}$.

Notăm $P_c(z) = z^2 + c, z, c \in C$; conform lui JULIA şi FATOU, J_c este **conexă dacă şi numai dacă** $0 \notin A(\infty)$, deci:

$$M = \{c \in C \mid P_c^k(0) \not\to \infty \text{ cînd } k \to \infty\}.$$

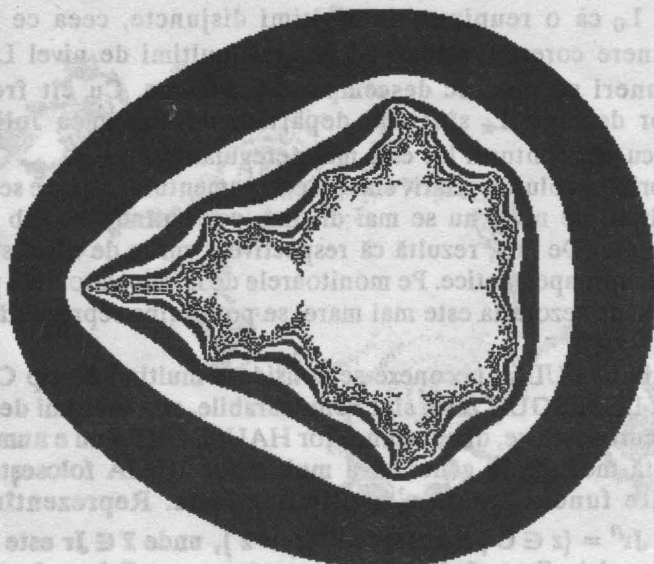
Această caracterizare se poate aplica pentru studii numerice.

5.4.1. Proprietăţi fundamentale ale mulţimii MANDELBROT

a) M este o mulţime conexă. Nu se cunoaşte dacă M este local conexă.

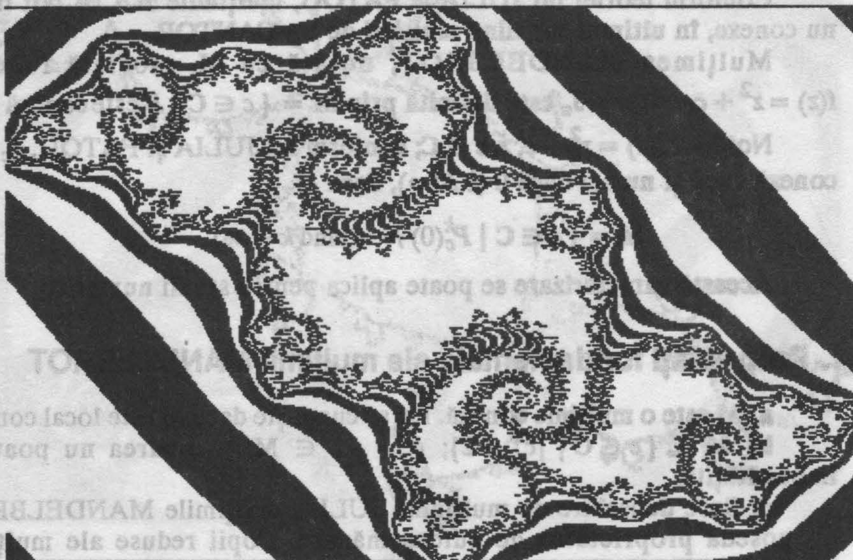
b) $M \subseteq \{c \in C \mid |c| \le 2\}$; $c = -2 \in M$; estimarea nu poate fi **îmbunătăţită.**

c) Spre deosebire de mulţimile JULIA, mulţimile MANDELBROT **nu posedă** proprietatea de autoasemănare. Copii reduse ale mulţimii MANDELBROT sînt legate de mulţimea mare prin nişte filamente al



Mulţimea B. MANDELBROT

Mulţimea G. JULIA



căror aspect depinde de poziţia copiilor.

d) Mulţimea MANDELBROT este fractală.

5.4.2. Clasificarea lui SULLIVAN a structurilor de mulţimi JULIA

a) Dacă c este în interiorul corpului principal al mulţimii MANDELBROT, care are forma unei cardioide, mulţimea JULIA se prezintă ca un cerc deformat fractal în jurul unui punct fix atractiv.

b) Dacă c este în interiorul unui „mugure" al mulţimii MANDELBROT, mulţimea JULIA constă dintr-o mulţime infinită de cercuri deformate fractal în jurul unui atractor periodic şi a preimaginilor lui.

c) Dacă c este în punctul de germinaţie al unui mugur, avem aşa-numitul caz parabolic: frontiera se prezintă asemeni unor cîrcei care se îmbogăţesc spre margine în raport cu un atractor stabil.

d) Dacă c este pe frontiera cardioidei sau a unui mugur avem cazul numit **discurile lui SIEGEL**, în interiorul regiunii limitate de mulţimea JULIA găsim cercuri invariante în jurul punctelor fixe.

e) Cazul numit **inelele lui HERMAN** nu are loc pentru $z \to z^2 + c$ şi nu a fost încă generat pe calculator.

5.4.3. Generarea pe calculator a mulţimii MANDELBROT.

Folosim proprietatea **b)** a mulţimii MANDELBROT. Dinamica punctelor este studiată tot cu ajutorul mulţimilor de nivel, aplicînd o metodă aproape identică cu BSM. Alegem o reţea de puncte $c \in C$ şi testăm pentru fiecare dacă după N iterări termenul corespunzător al şirului $0 \to c \to c^2 + c \to \ldots$ depăşeşte sau nu un disc centrat în origine de rază suficient de mare.

Analizînd desenul mulţimii MANDELBROT, observăm mai întîi o regiune delimitată de o cardioidă, simetrică faţă de o axa centrală cu adîncitura +0,25 şi cuprinsă la stînga pîna la -0,75. Urmează apoi un disc cu centrul în -1 şi raza 0,25, tangent la cardioidă. Mai observăm o mulţime practic infinită de discuri tangente la **M**, pentru ele există de asemenea o infinitate de discuri mici tangente. Mergînd mereu la stînga, pornind din cardioidă, se ajunge la punctul, numit MYRBERT - FEIGENBAUN, situat la -1,401. În acest punct se poate observa apariţia primei noi cardioide. Segmentul conţinut între acest punct şi -2 este conţinut în **M**. Pe acest segment se găsesc o infinitate de componente-cardioide, foarte mici. MANDELBROT a arătat că acestea sînt în număr infinit.

Dacă imaginăm un drum în c-plan care pleacă din **M** şi care se sfîrşeşte în exteriorul lui **M**, dacă îl variem pe c de-a lungul acestui drum, mulţimea JULIA asociată lui c prezintă o schimbare calitativă. Frontiera

lui M joacă rolul de fază de tranziție pentru funcția $z \to z^2 + c$.

5.5. Mulțimi de secțiune

Fie $f: \mathbb{C}^2 \to \mathbb{C}$, $f(z,c) = z^2 + c$. Considerăm f ca funcție reală:

$$f: \mathbb{R}^4 \to \mathbb{R}^2, \quad f(x,y,p,q) = (x^2 - y + p, 2xy + q)$$

Dacă fixăm p și q, $\mathrm{Jr} = \lim_{k \to \infty} \partial L_k(a)$, unde $L_k(a)$ sînt mulțimi de nivel, iar $L_0(a) \subseteq A(a)$ este fixată. Dacă fixăm $x = y = 0$, considerăm

$$M = \{ (p,q) \in \mathbb{R}^2 \mid P_c^k(0) \not\to \infty \text{ cînd } k \to \infty \},$$

și definim

$$L'_0(\infty) = \{ (p,q) \in \mathbb{R}^2 \mid p^2 + q^2 \geq 1/\varepsilon \}$$

$$L'_k(\infty) = \{ (p,q) \in \mathbb{R}^2 \mid P_c^k(x,y) \in L_0(\infty) \text{ și } P_c^l(x,y) \notin L_0(\infty), l < k \}$$

unde $\varepsilon \ll 1$, atunci $\partial M = \lim_{k \to \infty} \partial L'_k(\infty)$

După cum sublinia ADRIEN DOUDAY, mulțimile MANDELBROT și JULIA au fost obținute prin considerente de potențial care, intuitiv, este legat de așa-numitul „timp de evadare". Timpul de evadare al unui punct Z_0 al unei mulțimi JULIA sau MANDELBROT exprimă numărul N de iterări necesar pentru ca numărul Z_n obținut prin iterarea

$$Z_{n+1} = P_c(Z_n) = Z_n^2 + c$$

să aibă modulul mai mare decît un $r > 0$. Potențialul este aproximativ egal cu $\lg r/2^N$.

În general, pentru f și pentru $r > 0$ fixat vom considera funcția $T: \mathbb{R}^4 \to \mathbb{N} \cup \{+\infty\}$ (timpul de evadare),

$$T(x,y,p,q) = \begin{cases} \min \{ n \in \mathbb{N} \mid X_n^2 - Y_n^2 > r \}, & \text{dacă există } n \\ \infty, & \text{în caz contrar} \end{cases}$$

unde $X_{n+1} = X_n^2 - Y_n^2 + p$, $Y_{n+1} = 2 X_n Y_n + q$ și $X_0 = X$, $Y_0 = Y$.

Vom considera mulțimea $\tau = \{ (x,y,p,q) \in \mathbb{R}^4 \mid T(x,y,p,q) = \infty \}$. Mulțimile JULIA și MANDELBROT sînt situate în planul $\Pi^{p,q}(x,y)$ de secțiune prin τ determinat de p, q fixate, descris de x și de y și, respectiv în planul $\Pi^{0,0}(p,q)$ de secțiune prin τ determinat de $x=y=0$ și descris de p și q.

Potențialul punctelor lor se exprimă în funcție de $T \mid \Pi^{p,q}(x,y)$ și $T \mid \Pi^{0,0}(p,q)$.

Mulţimea τ se reprezintă deci în spaţiul cu 4 dimensiuni şi pentru a ne „apropia" de ea, trebuie să studiem secţiuni ale ei după diverse hiperplane, secţiuni care vor fi mulţimi tridimensionale. Deoarece, nici pe acestea nu le putem reprezenta convenabil pe calculator, ne vom mulţumi cu secţiuni plane prin acestea.

Am ales spre exemplu, hiperplanul determinat de condiţia $y=0$, şi x, p, q variabile. Considerăm mulţimea din spaţiulR^3:

$$\tau_{x,p,q} = \{(x,p,q) \mid T(x,0,p,q) = \infty\}$$

Secţionăm această mulţime după planele $\Pi(p,q)$, $\Pi(p,x)$, $\Pi(q,x)$. Secţiunea după primul dintre aceste plane determină mulţimea

$$\tau_{p,q} = \{ (p,q) \mid T(0,0,p,q) = +\infty\} \setminus \{ (p,q) \mid P^k_{p,q} \not\to \infty \text{ cînd } k \to \infty \} = M$$

deci, chiar mulţimea MANDELBROT conform celei de-a doua caracterizări.

Deci, mulţimea MANDELBROT este secţiunea prin τ după planul $\Pi^{0,0}(p,q)$.

Secţiunea după $\Pi^{q,0}(p,x)$ determină mulţimea

$$\tau_{x,p} = \{ (x,p) \mid T(x,0,p,q) = \infty, \text{unde } q \text{ e fixat} \},$$

deci, se poate defini mulţimea $\tau_{x,p}$ ca fiind:

$$\tau_{x,p} = \{ (p,x) \in R^2 \mid P^k_{0,q}(0,0) \not\to \infty, \text{cînd } k \to \infty \}$$

Experimental, distingem următoarele

Proprietăţi

a) Dacă $q=0$ mulţimea $\tau_{p,x}$ este conexă, altfel, în general. este neconexă.

b) $\tau_{x,p}$ intersectează $\tau_{p,q} = M$ după axa centrală OP.

$$\tau_{x,p} \subseteq \{ (x,p) \in R^2 \mid p^2 + x^2 \leq 2 \}$$

c) Dacă $q \in R$ astfel încît există $p \in R$ cu proprietatea $(p,q) \in M$, $\tau_{p,x}$ este nevidă. În caz contrar, planul de secţiune $\Pi^{0,q}(x,p)$ nu mai intersectează mulţimea MANDELBROT şi, aşa cum este de aşteptat, $\tau_{x,p}$ este vidă. Obţinem că pentru $|q| > 1{,}1$ mulţimea $\tau_{p,x}$ este vidă (vezi fig. 5-2).

Mulţimea $\tau_{x,p}$ ne furnizează informaţii despre mulţimea tip MANDELBROT pe care o obţinem dacă iterăm cu $y=0$ şi cu $x \neq 0$, adică:

$$M^* = \{ (p,q) \mid P^k(x,0) \not\to \infty \text{ cînd } k \to \infty, x \text{ fixat} \}$$

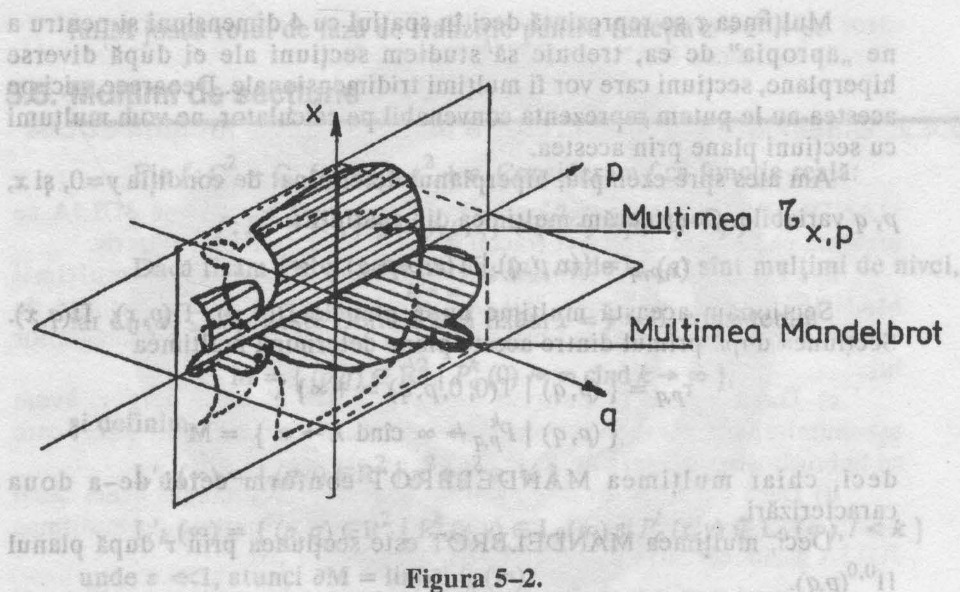


Figura 5–2.

Figura 5–3.

M^* coincide cu M dacă $x=0$, deci e conexă, şi în general neconexă dacă $x \neq 0$ (M^* taie mai multe „cozi de rîndunică"). De asemenea, $\tau_{x,p}$ este conexă dacă $\Pi^{p,q}(x,p)$ taie cardioida după axa centrală şi în general neconexă în caz contrar.

Observaţie: *Pentru $q \in \mathbf{R}$ fixat, observăm că $\tau_{x,p}$ taie mulţimile JULIA generate pentru $p=0$ şi $q \in \mathbf{R}$ după o axă „verticală" OX (vezi fig. 5–3). Acest lucru se poate vedea clar pentru mulţimea JULIA obţinută pentru $c=i$ (am folosit aici mulţimea JULIA, ca şi ADRIEN DOUDAY, în sensul de întreg desen reprezentat pe calculator).*

Secţiunea după $\Pi^{0,0}(x,q)$ determină mulţimea

$$\tau_{x,q} = \{ (x,q) \in \mathbf{R}^2 \mid T(x,0,p,q) = \infty, \text{ unde } p \text{ este fixat} \}$$

Deci, se poate defini mulţimea $\tau_{x,q}$ ca fiind

$$\tau_{x,q} = \{ (x,q) \in \mathbf{R}^2 \mid P^k_{0,p}(0,0) \not\to \infty, \text{ cînd } k \to \infty \}$$

Experimental, distingem următoarele

Proprietăţi

a) $\tau_{x,q}$ este totdeauna mulţime conexă (în caz că este mulţime nevidă).

b) $\tau_{x,q}$ intersectează $\tau_{p,q}=M$ după o axă paralelă cu axa OQ.

c) Dacă $p \in \mathbf{R}$ astfel încît există $q \in \mathbf{R}$ cu proprietatea $(p,q) \in M$, atunci mulţimea $\tau_{x,q}$ este nevidă. În caz contrar, planul de secţiune $\Pi^{0,p}(x,q)$ nu mai intersectează mulţimea MANDELBROT, deci, cum ar fi firesc, mulţimea $\tau_{x,q}$ este vidă (vezi fig. 5–4).

Pe baza secţiunilor după cele trei plane, ne putem imagina forma mulţimii spaţiale $\tau_{x,p,q}$ („meduza", vezi fig. 5–5).

Precizări:

1). *Am dori, în final să remarcăm complementaritatea informaţiilor pe care le aduc unele despre altele mulţimile de secţiune;*

2). *În cercetările pe plan mondial despre acest domeniu există relativ numeroase rezultate experimentale nejustificate teoretic. De aceea, recunoaştem caracterul preponderent empiric al contribuţiilor în acest moment, relativ la evidenţierea şi analiza acestor mulţimi de secţiune. Considerăm că ele pun într–o lumină nouă rezultatele cunoscute obţinute de către JULIA, MANDELBROT, DOUDAY şi ne permit să formulăm, în acest context, ca perspectivă de cercetare, iterarea funcţiilor (raţionale) n–dimensionale.*

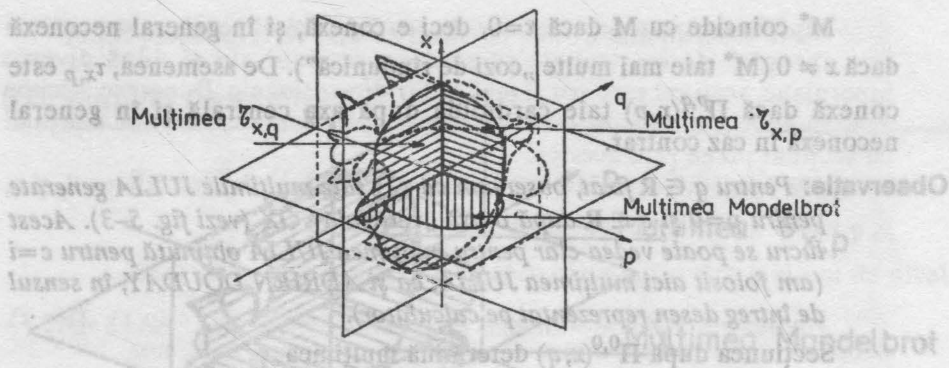


Figura 5–4.

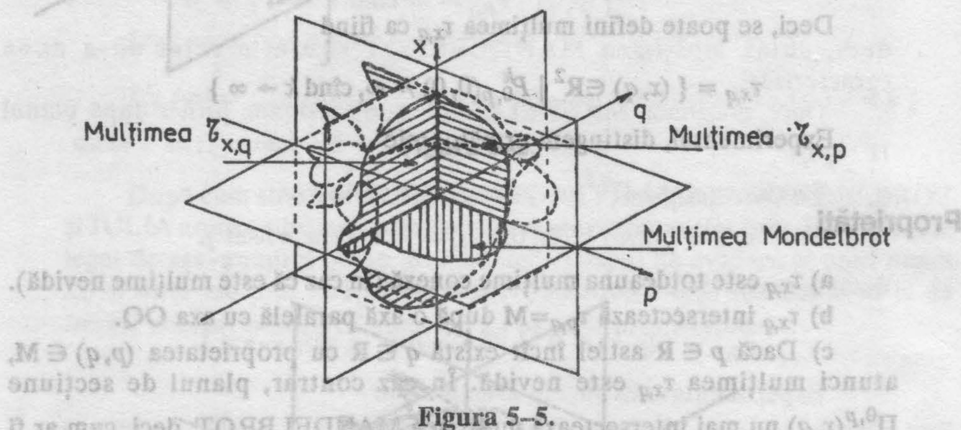


Figura 5–5.

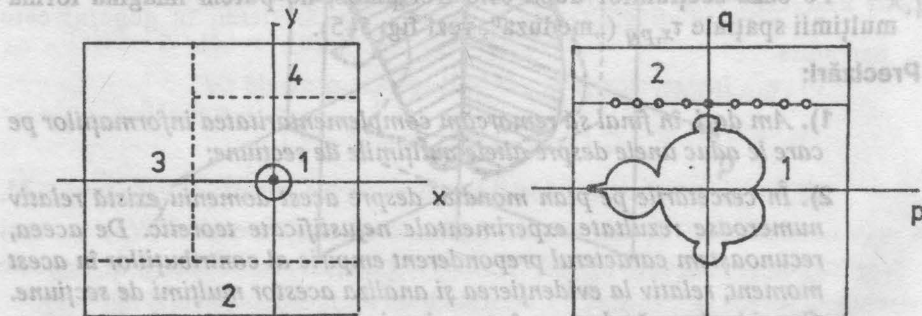


Figura 5–6.

Figura 5–7.

5.6. Proceduri şi programe grafice

Vom prezenta modurile de generare grafică a mulţimilor JULIA, MANDELBROT şi mulţimilor de secţiune cu ajutorul unor proceduri şi programe scrise în limbajul PASCAL–OREGON implementat pe minicalculatoare sub sistemul de operare RSX–11M.

Limbajul PASCAL–OREGON nu are implementate facilităţi grafice datorită dependenţei acestora de terminalul grafic aflat la dispoziţia utilizatorului. Aceste proceduri sînt specifice terminalelor DAF–2020 (alb–negru) şi DAF–2020C (color). Limbajul TURBO–PASCAL implementat pe microcalculatoare de tip IBM–PC sub sistemul de operare MS–DOS, are facilităţi grafice prin intermediul unei biblioteci de proceduri grafice. Vom observa că pentru optimizarea reprezentării grafice a mulţimilor de care ne ocupăm, în limbajul PASCAL–OREGON avem posibilitatea să elaborăm procedurile grafice astfel încît operaţiile de intrare/ieşire să utilizeze un timp de execuţie cît mai mic. În general, procedurile grafice se execută prin transmiterea unor coduri terminalului, care sînt interpretate prin trasare de linii, schimbarea fondului, schimbarea cernelii (pentru DAF–2020C).

Deoarece operaţiile de intrare/ieşire necesită mai mult timp decît celelalte operaţii, primele trebuie optimizate, folosindu–se cît mai rar posibil. Astfel, transmiterea unor coduri nu este recomandat să se facă cu ajutorul procedurii standard WRITE, deoarece scrierea unui număr de N caractere implică apelarea de N ori a procedurii de scriere a unui caracter. De aceea, s–a adoptat soluţia utilizării unui buffer de 256 de caractere, buffer în care se depun caracterele şi care sînt transmise spre terminal toate simultan, atunci cînd acesta e plin.

Procedurile ce utilizează acest buffer sînt:

INIBUF	iniţializează bufferul
OUTCHR(k: integer)	depune în buffer caracterul al cărui cod ASCII este k
OUTBUF	transmite bufferul la terminal

Bufferul este transmis spre terminal atunci cînd este folosită în mod explicit procedura `OUTBUF` sau, atunci cînd se apelează procedura `OUTCHR` încercîndu–se depunerea unui caracter, iar bufferul este plin, caz în care se transmite mai întîi bufferul spre terminal, după care bufferul este golit şi se depune în el caracterul.

Toate procedurile grafice care urmează folosesc `OUTCHR` în loc de `WRITE`.

Procedurile de grafică propriu–zise sînt următoarele:

MOVE(X, Y: integer)	poziționează punctul curent în punctul de coordonate absolute (X, Y).
DRAW(X, Y: integer)	trasează o linie începînd din punctul curent pînă în punctul de coordonate absolute (X, Y), punct care devine și punct curent.
APRINDE (X, Y: integer)	aprinde pe ecran punctul de coordonate (X, Y).

Observație: *Pentru DAF–2020C, atît la MOVE, cît și la DRAW avem:*

$0 \le X \le 447$ și $0 \le Y \le 287$,

iar pentru DAF–2020 avem :

$0 \le X \le 511$ și $0 \le Y \le 389$,

însă nu se pot vizualiza concomitent pe ecran mai mult de 288 linii cuprinse într–o fereastră, așa că se recomandă ca

$102 \le Y <= 389$.

Fereastra de vizualizare se poate modifica cu tastele–săgeți, astfel încît să se folosească, de exemplu, $0 \le Y \le 287$.

Alte proceduri grafice sînt:

INK(k: integer)	culoarea cernelii devine k
PAPER(k: integer)	culoarea fondului devine k (acestea două sînt valabile doar pentru DAF 2020C)
SUPERPOSE	afișarea se face cu supraimprimare
DELMODE	afișarea se face cu ștergere
INI_GR	șterge ecranul (devine verde, iar cerneala galbenă), afișarea se face cu ștergere, iar punctul grafic curent devine (0, 0).

Toate aceste proceduri se află în biblioteca GRAF.OLB (dedicată acestui capitol), și vor fi declarate în programele apelante folosind directiva `EXTERNAL`.

5.6.1. Desenarea mulțimilor JULIA

Pentru reprezentarea grafică a mulțimilor JULIA din planul XOY, se vor citi de la terminal următoarele valori: `p`, `q`, `lung`, `inalt`, `xmin`, `xmax`, `ymin`, `ymax`, unde:

- $c = p + i\,q$ este constantă complexă din procesul $z \rightarrow z^2 + c$ cu

condiţia $-2 \le$ `p,q` $\le +2$

- `lung` şi `inalt` reprezintă dimensiunile figurii de pe ecran, adică $0 \le$ `lung` ≤ 447 şi $0 \le$ `inalt` ≤ 287
- `xmin, xmax, ymin, ymax` reprezintă porţiunea din planul XOY ce se vizualizează pe ecran, adică $-2 \le$ `xmin,xmax` $\le +2$ şi $-2 \le$ `ymin,ymax` $\le +2$.

Pentru generarea grafică a mulţimilor JULIA vom folosi metoda BSM (Boundary Scanning Method) şi anume, pentru fiecare punct (`nx,ny`) de pe ecran, se determină punctul corespunzător (`x,y`) din planul XOY, determinîndu-se cît de repede diverge către infinit procesul $Z_{k+1} = Z_k^2 + c$ cu $Z_0 = (x,y)$, adică se determină valoarea k pentru care modulul numărului complex Z_k devine mai mare decît o valoare fixată `M`, de exemplu `M` = 100, sau dacă depăşeşte o anumită valoare `KMARE`, de exemplu `KMARE` = 160.

Deoarece, terminalul DAF-2020C poate afişa cel mult 8 culori, iar DAF-2020 cel mult 2 culori (alb şi negru), vom considera că `NRCULORI` este o constantă egală cu numărul de culori ce pot fi afişate de terminal şi vom colora cu aceeaşi culoare atît punctele ce evadează după k paşi cît şi cele ce evadează după k+`NRCULORI` paşi. Deoarece, există posibilitatea ca unele puncte să nu evadeze, unui punct care nu evadează după `KMAX` paşi i se va atribui culoarea neagră.

Astfel. desenarea figurii poate fi realizată cu ajutorul următoarelor proceduri:

```
procedure DESENEAZA_JULIA;
   var nx, ny, k: integer;
       deltax, deltay: real;
   begin
   deltax := (xmax - xmin) / lung;
   deltay := (ymax - ymin) / inalt;
   for ny := 0 to inalt do
      for nx := 0 to lung do
      begin
      CALCUL(xmin + nx * deltax,
             ymin + ny * deltay, k);
      ink(k);
      APRINDE(nx, ny)
      end
   end;
```

Procedurile `CALCUL` şi `APRINDE` sînt următoarele:

```
procedure CALCUL(x, y: real; var k: integer);
       var x1, y1: real;
   begin
```

```
            k := 0;
            repeat
              k := k + 1;
              x1 := sqr(x) - sqr(y) + p;
              y1 := 2 * x * y + q;
              x := x1;
              y := y1;
              r := sqr(x) + sqr(y)
            until (r > M) or (k >= KMARE);
            k := k mod NRCULORI
            end;
      procedure APRINDE ( x , y : integer ) ;
              begin
               move( x, y ) ;
               draw( x , y )
              end;
```

Valorile constantelor M, KMARE şi NRCULORI sînt declarate în programul principal apelant şi anume M = 100.0, KMARE = 160, NRCULORI = 8 pentru DAF–2020C şi NRCULORI = 2 pentru DAF–2020 (alb–negru).

Acest algoritm prezintă următoarele posibilităţi de optimizare:

- Mulţimea JULIA este simetrică faţă de origine, astfel că punctelor (x,y) şi (-x,-y) li se asociază aceeaşi culoare, deci desenarea mulţimii JULIA poate fi realizată în felul următor:

```
procedure DESENEAZA_JULIA;

        var nx, ny, k: integer;
        deltax, deltay: real;

     begin
     deltax := (xmax - xmin) / lung;
     deltay := (ymax - ymin) / inalt;
     for ny := 0 to inalt div 2 do
       for nx := 0 to lung do
       begin
       calcul(xmin + nx * deltax,
               ymin + ny * deltay, k);
       ink(k);
       aprinde(lung - nx , inalt - ny)
       end
     end;
```

- O optimizare posibilă a procedurii APRINDE ar fi să trasăm un vector doar atunci cînd se modifică culoarea, impunîndu–se astfel

introduccrea unor variabile suplimentare: xvechi, yvechi, kvechi.

- Deoarece, calculul culorii pentru fiecare pixel consumă mult timp, iar imaginile nu pot fi salvate aşa cum se realizează pe unele microcalculatoare, se pot crea fişiere în care să se introducă pentru fiecare punct culoarea asociată, iar la o desenare ulterioară a aceleaşi figuri, în loc să se calculeze culoarea, aceasta să fie citită din fişier.

Observaţie: *Optimizările prezentate reduc în mod spectaculos din timpul de desenare al unei figuri pe ecran, de la peste 60 minute, la mai puţin de 10 minute. Acest fapt atrage atenţia asupra modului (in)eficient de implementare pe minicalculatoare a bibliotecilor de grafică faţă de implementarile cunoscute pe microcalculatoare IBM–PC, dar şi asupra (in)dependenţei acestora de dispozitiv.*

5.6.2. Desenarea mulţimilor MANDELBROT

Pentru reprezentarea grafică a mulţimilor MANDELBROT se vor citi de la terminal următoarele valori: pmin, pmax, qmin, qmax, lung, inalt, unde:

- pmin, pmax, qmin, qmax reprezintă porţiunea din planul POQ ce se viziualizează pe ecran, cu $0 \le$ pmin,pmax≤ 447 şi $0\le$ qmin,qmax≤ 287;
- lung şi inalt reprezintă dimensiunile figurii de pe ecran .

Se va folosi o metodă aproape identică cu metoda BSM, şi anume se alege o reţea de puncte complexe c şi se testează pentru fiecare punct din reţea dacă după N iterări termenul corespunzător al şirului $0 \to c \to c^2 + c \to \ldots$ depăşeşte sau nu un disc centrat în origine şi de rază suficient de mare.

Se reprezintă mulţimea MANDELBROT din planul POQ situată în porţiunea:

$$\text{pmin} \le p \le \text{pmax şi qmin} \le q \le \text{qmax}.$$

Spre deosebire de generarea mulţimii JULIA, acum se iterează acelaşi proces $Z_{k+1} = Z_k^2 + c$, dar cu $Z_0 = (0,0)$ constant şi cu $c = \text{p} + i\,\text{q}$ variabil.

Procedurile de desenare şi calcul sînt :

```
procedure DESENEAZA_MANDELBROT;

     var np, nq, k:integer;
         deltap, deltaq: real;
```

```
        begin
        deltap := (pmax - pmin) / lung;
        deltaq := (qmax - qmin) / inalt;
        for nq := 0 to inalt do
          for np := 0 to lung do
          begin
          calcul(pmin + np * deltap,
                 qmin + nq * deltaq, k);
          ink(k);
          aprinde(np, nq)
          end
        end;
    procedure CALCUL(p, q: real; var k: integer);

          var x1, y1: real;

        begin
        k := 0;
          x := 0;
          y := 0;
        repeat
          k := k + 1;
          x1 := sqr(x) - sqr(y) + p;
          y1 := 2 * x * y + q;
          x := x1;
          y := y1;
          r := sqr(x) + sqr(y)
        until (r > M) or (k >= KMARE);
        k := k mod NRCULORI
        end;
```

Mulţimea Mandelbrot nu este simetrică faţă de origine dar este simetrică faţă de axa OP deoarece:

- pentru $(p,q):(0,0)\to(p,q)\to(p^2-q^2+p,2pq+q)\to\ldots$
- pentru $(p,-q):(0,0)\to(p,-q)\to(p^2+q^2+p,-2pq-q)\to\ldots$

Modulele termenilor şirurilor sînt egale, deci, un termen al primului şir depăşeşte în modul valoarea M atunci cînd un termen al celui de-al doilea şir depăşeşte în modul valoarea M.

Rutina de desenare în cazul qmin = -qmax este următoarea:

```
    procedure Deseneaza_Mandelbrot;

        var np, nq, k:integer;
            deltap, deltaq: real;
```

```
begin
deltap := (pmax - pmin) / lung;
deltaq := (qmax - qmin) / inalt;
for nq := 0 to inalt div 2 do
    for np := 0 to lung do
    begin
    calcul(pmin + np * deltap,
           min + nq * deltaq, k);
    ink(k);
    aprinde(np, nq);
    aprinde(np, inalt - nq);
    end
end;
```

Observaţii:

1. *În cazul cînd „ferestrele" de reprezentare pentru mulţimile JULIA şi MANDELBROT sînt dreptunghiuri necentrate în origine, pentru a optimiza algoritmii se poate proceda astfel: pentru o mulţime JULIA, se desenează zona 1 prin simetrie, iar zonele 2, 3 şi 4 fără simetrie; pentru mulţimea MANDELBROT, întîi se desenează zona 2 fără simetrie (vezi fig. 5–6 şi fig. 5–7).*

2. *Pentru a studia dinamica formei cînd c se apropie de frontiera mulţimii MANDELBROT, este util să cunoaştem valori ale lui c în anumite puncte „speciale": puncte de germinaţie ale mugurilor, punctul de adîncitură al cardioidei etc.*

Pentru aceasta ne folosim de utilitatea furnizată de terminalele grafice menţionate care, în regim „introducere grafică", afişează un cursor cruce pe care îl putem deplasa pe ecran cu ajutorul săgeţilor. Din acest regim se iese la apăsarea unei taste oarecare, diferită de săgeţi, iar la ieşirea din acest regim, terminalul ne transmite codul tastei şi, ceea ce este mai important, coordonatele pe care se află cursorul cruce. Făcînd anumite calcule simple se poate afla valoarea *c* corespunzătoare punctului de pe ecran.

5.6.3. Desenarea mulţimilor de secţiune

Pentru reprezentarea grafică a mulţimii $\tau_{x,p}$ se citesc de la terminal următoarele: `lung, inalt, q, pmin, pmax, xmin, xmax`, iar pentru mulţimea $\tau_{x,q}$: `lung, inalt, p, qmin, qmax, xmin, xmax`.

Figurile se desenează analog ca şi mulţimea Mandelbrot, doar că se înlocuieşte unde este cazul q cu x, respectiv p cu x, iar în rutina de calcul nu se modifică decît variabilele corespunzătoare.

Se observă că mulţimea $\tau_{x,p}$ este simetrică faţă de axa orizontală OP:

- (p,x): $(x,0) \rightarrow (x^2+p,q) \rightarrow \ldots$
- $(p,-x)$: $(-x,0) \rightarrow (x^2+p,q) \rightarrow \ldots$

Se observă că mulţimea $\tau_{x,q}$ este simetrică atît faţă de axa orizontală OQ cît şi faţă de axa verticală OX:

- (q,x): $(x,0) \rightarrow (x^2+p,q) \rightarrow \ldots$
- $(q,-x)$: $(-x,0) \rightarrow (x^2+p,q) \rightarrow \ldots$

şi respectiv:

- (q,x): $(x,0) \rightarrow (x^2+p,q) \rightarrow \ldots$
- $(-q,x)$: $(x,0) \rightarrow (x^2+p,-q) \rightarrow \ldots$

deoarece termenii corespunzători celor două şiruri evadează la ∞ cu aceeaşi viteză.

5.7. Pachetul de programe FRACTALI

5.7.1. Prezentare generală

Pentru implementarea celor prezentate mai sus, s–au elaborat 9 programe în limbajul PASCAL–OREGON sub sistemul de operare RSX–11M, care să realizeze reprezentarea grafică a mulţimilor JULIA şi MANDELBROT conform metodelor prezentate mai sus. Aceste programe utilizează procedurile grafice din biblioteca GRAF.olb pe care o vom prezenta la sfîrşitul capitolului (secţiunea 5.7.4). Reprezentarea grafică a acestor mulţimi se realizează pe terminalul grafic color DAF–2020C, dar şi pe terminalul grafic alb–negru DAF–2020, de asemenea prin copierea ecranului, desenul poate fi reprodus pe imprimanta matricială. Pachetul de programe FRACTALI are următoarele componente :

- programe care să realizeze reprezentarea grafică direct pe terminalul grafic utilizat ;
- programe care să realizeze reprezentarea grafică pe terminal prin utilizarea la intrare a unor fişiere intermediare create anterior;
- programe pentru crearea de fişiere intermediare în scopul utilizării lor ulterioare la reprezentarea grafică pe terminal.

Programele care sînt componente ale pachetului sînt: JUL, MAN, JJ, SJ, SM, CJ, CM, DCJ, DCM. Funcţiile pachetului de programe sînt clasificate astfel:

a) reprezentarea grafică direct pe terminalul grafic color DAF–2020C, ce se realizează cu următoarele programe :

- JUL.pas; desenarea mulţimilor JULIA
- MAN.pas; desenarea mulţimilor MANDELBROT

Exemple de utilizare

```
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
... ... ...
>run MAN
Introduceti lung inalt pmin pmax qmin qmax
... ... ...
```

b) reprezentarea grafică direct pe terminalul grafic alb–negru DAF–2020, ce se realizează cu programul JJ.pas;

Exemplu de utilizare

```
>run JJ
Introduceti, in ordine: p,q,lung,inalt,xmin,xmax,ymin,ymax
... ... ...
```

c) crearea de fişiere intermediare pentru utilizarea ulterioară la reprezentarea grafică pe terminal, aceasta realizîndu-se cu următoarele programe:

- SJ.pas; pentru mulţimile JULIA
- SM.pas; pentru mulţimile MANDELBROT

Exemple de utilizare

```
>run SJ
Numele fisierului:
Introduceti, in ordine: p,q,lung,inalt,xmin,xmax,ymin,ymax
... ... ...
>run SM
Numele fisierului:
Introduceti lung inalt pmin pmax qmin qmax
... ... ...
```

d) reprezentarea grafică pe terminalul grafic color DAF–2020C folosind fişiere intermediare, aceasta se realizează cu următoarele programe :

- CJ.pas; pentru mulţimile JULIA
- CM.pas; pentru mulţimile MANDELBROT

Exemple de utilizare

```
>run CJ
Numele fisierului :
... ... ...
>run CM
Numele fisierului :
... ... ...
```

e) reprezentarea grafică pe terminalul grafic alb–negru DAF–2020 folosind fişiere intermediare, aceasta realizîndu-se cu următoarele programe:

- DCJ,pas; pentru mulțimile JULIA
- DCM.pas; pentru mulțimile MANDELBROT

Exemple de utilizare

```
>run DCJ
Numele fisierului:
... ... ...
>run DCM
Numele fisierului:
... ... ...
```

Precizăm faptul că pentru realizarea formei executabile a unui program din cele de mai sus, trebuie ca în comanda destinată link–editării, pe lîngă indicarea bibliotecii PASLIB, să se indice și biblioteca GRAF ce conține procedurile grafice și care execută reprezentări grafice pe terminal. Astfel, de exemplu, pentru programul JUL comenzile de operare vor fi următoarele:

```
>pas JUL=JUL
>tkb JUL/fp/cp=JUL,[1,1]GRAF/lb,lb:[1,1]PASLIB/lb
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
-0.18 0.667 447 287 -1.5 1.5 -1.5 1.5
```

Pentru testarea programului JUL precizăm cîteva variante în funcție de valorile parametrilor p și q ce determină obținerea diferitelor forme ale mulțimii JULIA :

```
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
0.27334 0.007421 447 287 -2 2 -2 2
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
-0.74543 0.113011 447 287 -2 2 -2 2
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
-0.194 0.65571 447 287 -2 2 -2 2
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
0.11031 -0.67037 447 287 -2 2 -2 2
>run JUL
Introduceti p q lung inalt xmin xmax ymin ymax
-0.39054 -0.58679 447 287 -2 2 -2 2
```

Pentru testarea programului MAN ce realizează reprezentarea grafică a mulțimii MANDELBROT, se poate proceda astfel:

```
>run MAN
Introduceti lung inalt pmin pmax qmin qmax
447 287 -1.5 1.5 -1.5 1.5
```

Pentru utilizarea pachetului de programe FRACTALI se poate apela procedura de comenzi indirecte FRACTALI.cmd ce va fi prezentată în cele ce urmează.

5.7.2. Procedura de comenzi indirecte FRACTALI.cmd

```
.disable display
.10;
;              *****   F R A C T A L I   *****
;___________________________________________________________
; Pachet de programe pentru reprezentarea grafica
; a multimilor JULIA si MANDELBROT pe terminalele grafice
; color sau,alb-negru , desenarea realizindu-se direct pe
; terminal sau prin intermediul unor fisiere intermediare
; create anterior
; AUTORI : M. VLADA, std. S. BRINZEI, std. M. Gidea
;___________________________________________________________
;    OBSERVATIE : pentru transmiterea desenului de pe
; terminal pe imprimanta matriciala, se testeaza :
;   <CTRL> + <PF2> + <S> ( <S> pentru copiere simpla,
;   respectiv <D> pentru copiere dubla )
;___________________________________________________________
.START:
;           FRACTALI : OPTIUNI pentru utilizare
;   1 = desenarea directa pe terminalul color
;   2 = desenarea directa pe terminalul alb-negru
;   3 = crearea de fisiere intermediare in vederea desenarii
;   4 = desenarea pe terminalul color folosind fis. intermed.
;   5 = desenarea pe terminalul alb-negru folosind fisiere
;       intermediare
;   6 = EXIT
;-----------------------------------------------------------
.20:
.askn OPT tastati o optiune ( 1,2,3,4,5,6) ;
.if OPT eq 6 .goto STOP
.if OPT eq 5 .goto 140
.if OPT eq 4 .goto 110
.if OPT eq 3 .goto 70
.if OPT eq 2 .goto 60
.if OPT eq 1 .goto 30
.goto 20
.30:
;-----------------------------------------------------------
;Desenarea DIRECTA a multimilor JULIA si MANDENBROT
;OPTIUNI :1 = multimi JULIA, 2 =multimi MANDELBROT, 3=EXIT
.askn OPT tastati optiunea ( 1,2,3 ) :
.if OPT eq 3 .goto START
.if OPT eq 2 .goto 50
.if OPT eq 1 .goto 40
.goto 30
.40:
;Desenarea DIRECTA pe terminalul color a multimii JULIA
```

```
run JUL
.wait JUL
.goto START
.50:
;Desenarea DIRECTA pe terminalul color a multimii MANDELBROT
run MAN
.wait MAN
.goto START
.60:
;-----------------------------------------------------------
;Desenarea DIRECTA pe terminalul alb-negru a multimii JULIA
run JJ
.wait JJ
.goto START
.70:
;-----------------------------------------------------------
;Crearea de fisiere intermediare pentru desenare
;OPTIUNI : 1=multimi JULIA, 2=multimi MANDELBROT, 3=EXIT
.askn OPT Tastati optiunea ( 1,2,3 ) :
.if OPT eq 3 .goto START
.if OPT eq 2 .goto 90
.if OPT eq 1 .goto 80
.goto 70
.80:
;Crearea de fisiere intermediare pentru multimea JULIA
run SJ
.wait SJ
.goto 100
.90:
;Crearea de fisiere intermediare pentru multimi MANDELBROT
run SM
.wait SM
.100:
;Desenarea pe terminalul color sau alb-negru
;OPTIUNI : 1= terminal color, 2= terminal alb-negru, 3=EXIT
.askn OPT Tastati optiunea ( 1,2,3 ) ;
.if OPT eq 3 .goto START
.if OPT eq 2 .goto 140
.if OPT eq 1 .goto 110
.goto 100
.110:
;-----------------------------------------------------------
;Desenarea pe terminalul color folosind fisiere intermediare
;OPTIUNI : 1=multimi JULIA, 2=multimi MANDELBROT, 3 = EXIT
.askn OPT Tastati optiunea ( 1,2,3 ) :
.if OPT eq 3 .goto START
.if OPT eq 2 .goto 130
.if OPT eq 1 .goto 120
.goto 110
.120:
run CJ
.wait CJ
```

```
.goto START
.130:
run CM
.wait CM
.goto START
.140:
;---------------------------------------------------------------
;Desenarea pe terminalul alb-negru folosind fisiere intermediare
;OPTIUNI : 1=multimi JULIA, 2=multimi MANDELBROT, 3 =EXIT
.askn OPT Tastati optiunea ( 1,2,3 ) :
.if OPT eq 3 .goto START
.if OPT eq 2 .goto 160
.if OPT eq 1 .goto 150
.goto 140
.150:
run DCJ
.wait DCJ
.goto START
.160:
run DCM
.wait DCM
.goto START
.STOP:
.exit
```

5.7.3. Programe sursă

Programul JUL.pas

```
{[l-]}
{$NOCHECK}
program JUL;
{ desenarea directa pe terminalul color
  a multimilor JULIA   }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
    lungime_ecran = 447;
  type
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimx, minimy, x, deltax, deltay, p, q, xmin, xmax,
    ymin, ymax, y, x1, x2, y2, r: real;
    dr, sscx1, sscy1, scx1, scx2, scy1, scy2, inalt, lung,
    nx, ny, k, kvechi , xvechi, yvechi: integer;
  procedure inibuf;  external;
  procedure delmode; external;
```

```
 procedure outbuf;  external;
 procedure move(x, y: integer); external;
 procedure draw(x, y: integer); external;
 procedure ink(k: integer); external;
%include aprinde.pas;
%include calcul.pas;
 procedure bipunct(nx, ny, k: integer);
   begin
     if nx = 0 then
       begin
       kvechi := k;
       xvechi := nx;
       yvechi := ny;
       end
     else if (k <> kvechi) or (nx = dr) then
       begin
       ink(redef[kvechi]);
       move(scx1 + xvechi,scy1 + yvechi);
       draw(scx1 + nx,scy1 + ny);
       move(scx2 - xvechi,scy2 - yvechi);
       draw(scx2 - nx,scy2 - ny);
       if (k <> kvechi) and (nx = dr) then
         begin
         ink(redef[k]);
         aprinde(scx1 + nx,scy1 + ny);
         aprinde(scx2 - nx,scy2 - ny);
         end;
       kvechi := k;
       xvechi := nx;
       yvechi := ny
       end;
   end;
 procedure punct(nx, ny, k: integer);
   begin
     if nx = 0 then
       begin
       kvechi := k;
       move( scx1, scy1 + ny);
       end
     else if (k <> kvechi) or (nx = dr) then
       begin
       ink(redef[kvechi]);
       draw(scx1 + nx,scy1 + ny);
       if (k <> kvechi) and (nx = dr) then
         begin
         ink(redef[k]);
         draw(scx1 + nx,scy1 + ny);
         end;
       kvechi := k;
       end;
   end;
```

```
%include julia1.pas;

  procedure scalare;
    begin
      sscx1 := lungime_ecran div 2 + round(xmin / deltax);
      if sscx1 < 0 then
        sscx1 := 0
      else if sscx1 > lungime_ecran - lung then
        sscx1 := lungime_ecran - lung;
      sscy1 := inaltime_ecran div 2 + round(ymin / deltay);
      if sscy1 < 0 then
        sscy1 := 0
      else if sscy1 > inaltime_ecran - inalt then
        sscy1 := inaltime_ecran - inalt;
    end; {SCALARE}
  procedure deseneaza_figura;
    begin
      inibuf;
      delmode;
      verifica_datele;
      deltax := (xmax - xmin) / lung;
      deltay := (ymax - ymin) / inalt;
      scalare;
      min( - xmin, xmax, - ymin, ymax, minimx, minimy);
      if (minimx <= 0) or (minimy <= 0) then
        dreptunghi(xmin, xmax, ymin, ymax)
      else
        begin
        patrat(minimx, minimy);
        dreptunghi(xmin, minimx, ymin, - minimy);
        dreptunghi(xmin, - minimx, - minimy, ymax);
        dreptunghi( - minimx, xmax, minimy, ymax);
        dreptunghi(minimx, xmax, ymin, minimy);
        end;
      outbuf;
    end;

  begin { main }
    writeln('Introduceti p q lung inalt xmin xmax ymin ymax ');
    read(p, q, lung, inalt, xmin, xmax, ymin, ymax);
    deseneaza_figura;
  end.
```

Programul MAN.pas

```
{[l-]}
{$NOCHECK}
program MAN;
{ desenarea directa pe terminalul color
  a multimilor MANDELBROT }
  const
```

```
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
    lungime_ecran = 447;
  type
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimp, minimq, x, deltap, deltaq, p, q, pmin, pmax,
    qmin, qmax, y, x1, x2, y2, r: real;
    dr, sscp1, sscq1, scp1, scp2, scq1, scq2, inalt, lung,
    np, nq, k, kvech , pvechi, qvechi: integer; ch: char;
  procedure inibuf; external;
  procedure delmode; external;
  procedure outbuf; external;
  procedure move(x, y: integer); external;
  procedure draw(x, y: integer); external;
  procedure ink(k: integer); external;
  procedure pozc(var x, y: integer;
                 var ch: char); external;
%include aprinde.pas;

  procedure punct(np, nq, k: integer);
    begin
      if np = 0 then
        begin
        kvechi := k;
        move(scp1, scq1 + nq);
        end
      else if (k <> kvechi) or (np = dr) then
        begin
        ink(redef[kvechi]);
        draw(scp1 + np, scq1 + nq);
        if (k <> kvechi) and (np = dr) then
          begin
          ink(redef[k]);
          draw(scp1 + np, scq1 + nq);
          end;
        kvechi := k;
        end;
    end;
  procedure scalare;
    begin
      sscp1 := lungime_ecran div 2 + round(pmin / deltap);
      if sscp1 < 0 then
        sscp1 := 0
      else if sscp1 > lungime_ecran - lung then
        sscp1 := lungime_ecran - lung;
      sscq1 := inaltime_ecran div 2 + round(qmin / deltaq);
      if sscq1 < 0 then
```

```
      sscql := 0
    else if sscql > inaltime_ecran - inalt then
      sscql := inaltime_ecran - inalt;
  end; {SCALARE}
procedure bipunct(np, nq, k: integer);
  begin
    if np = 0 then
      begin
      kvechi := k;
      pvechi := np;
      qvechi := nq;
      end
    else if (k <> kvechi) or (np = dr) then
      begin
      ink(redef[kvechi]);
      move(scpl + pvechi, scql + qvechi);
      draw(scpl + np, scql + nq);
      move(scpl + pvechi, scq2 - qvechi);
      draw(scpl + np, scq2 - nq);
      pvechi := np;
      qvechi := nq;
      if (k <> kvechi) and (np = dr) then
        begin
        ink(redef[k]);
        aprinde(scpl + np, scql + nq);
        aprinde(scpl + np, scq2 - nq);
        end;
      kvechi := k;
      end;
  end;

%include mandelbl.pas;

{ in DREPTUNGHI se inlocuieste PUNE cu PUNCT;
  in PATRAT se inlocuieste PUNE cu BIPUNCT}
  procedure deseneaza_figura;
    begin
      verifica_datele;
      deltap := (pmax - pmin) / lung;
      deltaq := (qmax - qmin) / inalt;
      scalare;
      min( - qmin, qmax, minimq);
      if minimq <= 0 then
        dreptunghi(pmin, pmax, qmin, qmax)
      else
        begin
        patrat(pmin, pmax, minimq);
        dreptunghi(pmin, pmax, qmin, - minimq);
        dreptunghi(pmin, pmax, minimq, qmax);
        end;
      outbuf;
    end;
  procedure afla_coord(var p, q: real;
```

```
                       var ch: char);
  begin
    pozc(np, nq, ch);
    p := (np - sscpl) * deltap + pmin;
    q := (nq - sscql) * deltaq + qmin;
  end;

begin { main }
  writeln('Introduceti lung inalt pmin pmax qmin qmax ');
  read(lung, inalt, pmin, pmax, qmin, qmax);
  deseneaza_figura;
  repeat;
    afla_coord(p, q, ch);
    writeln('P=', p: 8: 6, 'Q=', q: 8: 6)
  until ord(ch) = 13;
end.
```

Programul JJ.pas

```
{[l-]}
{$OWN}
program jj;
{ desenarea directa pe terminalul alb-negru
  a multimilor JULIA  }
  const
    inaltime_ecran = 287;
    lungime_ecran = 447;
    kmare = 20;
    txt_len = 6;
  type
    pointer = ^nod;
    nod =
      record
        absc, ordo: real;
        leg: pointer
      end;
    bit = 0..1;
  var
    ecran: packed array [0..lungime_ecran, 0..inaltime_ecran]
          of bit;
    deltax, deltay, p, q, xmin, xmax, ymin, ymax, x, y: real;
    sscxl, sscyl, inalt, lung, nx, ny: integer;
    urm, cap, coada, sfirsit: pointer;
    gata: boolean;

    %include outti.def;
    %include tekdr.def;

  procedure calc_coord(x, y: real);
    begin
      nx := round((x - xmin) / deltax);
      ny := round((y - ymin) / deltay);
```

```
    end;
  procedure creaza(x, y: real);
    var
      z: pointer;
    begin
      new(z);
      with z^ do
        begin
        absc := x;
        ordo := y;
        leg := nil
        end;
      sfirsit^.leg := z;
      sfirsit := z
    end;
  procedure testeaza(x, y: real);
    begin
      calc_coord(x, y);
      if (0 <= nx) and (nx <= lung) and (0 <= ny)
        and (ny <= inalt) then
        if (ecran[nx, ny] = 0) then
          begin
          ecran[nx, ny] := 1;
          creaza(x, y);
          gata := false;
          aprinde(sscx1 + nx, sscy1 + ny)
          end
        else
      else
        creaza(x, y)
    end;
  procedure calc_invers(a, b: real;
                        var x, y: real);
    var
      z: real;
    begin
      if b = 0 then
        if a >= 0 then
          begin
          x := sqrt(a);
          y := 0
          end
        else
          begin
          x := 0;
          y := sqrt( - a)
          end
      else
        begin
        z := sqrt(sqr(a) + sqr(b));
        x := sqrt((z + a) / 2);
        y := b / (2 * x)
        end
```

```
  end;
procedure inverseaza(x, y: real);
  var
    x1, y1: real;
  begin
    calc_invers(x - p, y - q, x1, y1);
    testeaza(x1, y1);
    if (x1 <> 0) or (y1 <> 0) then
      testeaza( - x1, - y1)
  end;
procedure taie_capul;
  var
    urm: pointer;
  begin
    urm := cap^.leg;
    dispose(cap);
    cap := urm
  end;
procedure scalare;
  begin
    deltax := (xmax - xmin) / lung;
    deltay := (ymax - ymin) / inalt;
    sscx1 := lungime_ecran div 2 + round(xmin / deltax);
    if sscx1 < 0 then
      sscx1 := 0
    else if sscx1 > lungime_ecran - lung then
      sscx1 := lungime_ecran - lung;
    sscy1 := inaltime_ecran div 2 + round(ymin / deltay);
    if sscy1 < 0 then
      sscy1 := 0
    else if sscy1 > inaltime_ecran - inalt then
      sscy1 := inaltime_ecran - inalt;
  end; {SCALARE}
procedure verifica_datele;
  procedure schimba(var x, y: real);
    var
      z: real;
    begin
      z := x;
      x := y;
      y := z
    end;
  begin
    if xmin > xmax then
      schimba(xmin, xmax);
    if ymin > ymax then
      schimba(ymin, ymax);
    if lung > lungime_ecran then
      lung := lungime_ecran;
    if inalt > inaltime_ecran then
      inalt := inaltime_ecran
  end;
procedure deseneaza_figura;
```

```
    var
      i: integer;
    begin
      verifica_datele;
      scalare;
      ecran[0, 0] := 0;
      new(cap);
      sfirsit := cap;
      calc_invers(0.25 - p, - q, x, y);
      testeaza(0.5 + x, y);
      if (x <> 0) or (y <> 0) then
        testeaza(0.5 - x, - y);
      taie_capul;
      coada := sfirsit;
      repeat
        gata := true;
        while cap <> coada do
          begin
          inverseaza(cap^.absc, cap^.ordo);
          taie_capul
          end;
          inverseaza(cap^.absc, cap^.ordo);
        coada := sfirsit
      until gata;
    end;
  procedure citeste_datele;
    begin
      writeln('Introduceti,in ordine : p,q,lung,inalt,
              xmin,xmax,ymin,ymax');
      readln(p, q, lung, inalt, xmin, xmax, ymin, ymax);
    end;

  begin { main }
    inibuf;
    ini_gr;
    citeste_datele;
    deseneaza_figura;
    outbuf;
  end.
```

Programul SJ.pas

```
{[l-]}
{$NOCHECK}
program SJ;
{ crearea de fisiere intermediare pentru
  desenarea multimilor JULIA  }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
```

```
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimx, minimy, x, deltax, deltay, p, q, xmin, xmax,
    ymin, ymax, y, x1, x2, y2, r: real;
    opt, dr, sscx1, sscy1, scx1, scx2, scy1, scy2, inalt, lung,
    nx, ny, k, kvechi, xvechi, yvechi: integer;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure detach; external;
  procedure scrie;
    begin
      write(fisier, inreg);
      break(fisier);
      inreg.nr := 0
    end;
  procedure pune(ch: char);
    begin
      with inreg do
        begin
        buf[nr] := ch;
      if nr >= lgbuf then
          scrie
        else
          nr := succ(nr);
        end
    end;
  procedure conv_real(var x: real;
                      var nr: integer;
                      var f: buffer); external;
  procedure conv_int(var x: integer;
                     var nr: integer;
                     var f: buffer);  external;
  procedure converteste;
    begin {CONVERTESTE}
      with inreg do
        begin
        conv_real(p, nr, buf);
        conv_real(q, nr, buf);
        conv_int(lung, nr, buf);
        conv_int(inalt, nr, buf);
```

```
        conv_real(xmin, nr, buf);
        conv_real(xmax, nr, buf);
        conv_real(ymin, nr, buf);
        conv_real(ymax, nr, buf);
        end
    end;

%include calcul.pas;

  procedure punefis(x, y, k: integer);
    begin
      pune(chr(k))
    end;

%include julia1.pas;

{ in DREPTUNGHI se inlocuieste PUNCT cu PUNE
  in PATRAT se inlocuieste BIPUNCT cu PUNE}
  procedure deseneaza_figura;
    begin
      deltax := (xmax - xmin) / lung;
      deltay := (ymax - ymin) / inalt;
    min( - xmin, xmax, - ymin, ymax, minimx, minimy);
      if (minimx <= 0) or (minimy <= 0) then
        dreptunghi(xmin, xmax, ymin, ymax)
      else
        begin
        patrat(minimx, minimy);
        dreptunghi(xmin, minimx, ymin, - minimy);
        dreptunghi(xmin, - minimx, - minimy, ymax);
        dreptunghi( - minimx, xmax, minimy, ymax);
        dreptunghi(minimx, xmax, ymin, minimy);
        end;
    end;
  procedure citeste_datele;
    begin
      writeln('Introduceti,in ordine : p,q,lung,inalt,xmin,
               xmax,ymin,ymax');
      readln(p, q, lung, inalt, xmin, xmax, ymin, ymax);
    end;
  procedure citnumefis;
    begin
      write('Numele fisierului :');
      readln(numefis);
    end;

  begin { main }
    citnumefis;
    rewrite(fisier, numefis);
    citeste_datele;
      verifica_datele;
           detach;
```

```
    converteste;
    deseneaza_figura;
    scrie;
    break (fisier);
  end.
```

Programul SM.pas

```
{[l-]}
{$NOCHECK}
program SM;
{ crearea de fisiere intermediare pentru
  desenarea multimilor MANDELBROT  }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimp, minimq, x, deltap, deltaq, p, q, pmin, pmax,
    qmin, qmax, y, x1, x2, y2, r: real;
    dr, sscp1, sscq1, scp1, scp2, scq1, scq2, inalt, lung,
    np, nq, k, kvechi, pvechi, qvechi: integer;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure detach; external;
  procedure scrie;
    begin
      write(fisier, inreg);
      inreg.nr := 0
    end;
  procedure pune(ch: char);
    begin
      with inreg do
        begin
        buf[nr] := ch;
        if nr >= lgbuf then
          scrie
```

```
        else
          nr := succ(nr);
        end
    end;
  procedure conv_real(var x: real;
                      var nr: integer;
                      var f: buffer); external;
  procedure conv_int(var x: integer;
                     var nr: integer;
                     var f: buffer);  external;
  procedure converteste;
    begin {CONVERTESTE}
      with inreg do
        begin
        conv_int(lung, nr, buf);
        conv_int(inalt, nr, buf);
        conv_real(pmin, nr, buf);
        conv_real(pmax, nr, buf);
        conv_real(qmin, nr, buf);
        conv_real(qmax, nr, buf);
        end
    end;
  procedure citnumefis;
    begin
      write('Numele fisierului :');
      readln(numefis);
    end;

%include mandelbl.pas;

  procedure deseneaza_figura;
    begin
      verifica_datele;
      deltap := (pmax - pmin) / lung;
      deltaq := (qmax - qmin) / inalt;
      min( - qmin, qmax, minimq);
      if minimq <= 0 then
        dreptunghi(pmin, pmax, qmin, qmax)
      else
        begin
        patrat(pmin, pmax, minimq);
        dreptunghi(pmin, pmax, qmin, - minimq);
        dreptunghi(pmin, pmax, minimq, qmax)
        end;
    end;
  procedure citeste_datele;
    begin
      writeln('Introduceti lung inalt pmin pmax qmin qmax');
      read(lung, inalt, pmin, pmax, qmin, qmax)
    end;


  begin { main }
```

```
    citnumefis;
    rewrite(fisier, numefis);
    citeste_datele;
    verifica_datele;
    inreg.nr := 0;
    detach;
    converteste;
    deseneaza_figura;
    scrie;
    break(output)
  end.
```

Programul CJ.pas

```
{[l-]}
{$NOCHECK}
program CJ;
{ desenarea pe terminalul color a multimilor
  JULIA folosind fisiere intermediare  }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
    us = 31;
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimx, minimy, x, deltax, deltay, p, q, xmin, xmax,
    ymin, ymax, y, x1, x2, y2, r: real;
    opt, dr, sscx1, sscy1, scx1, scx2, scy1, scy2, inalt,
    lung, nx, ny, k, kvechi, xvechi, yvechi: integer;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure inibuf;  external;
  procedure outchr(k: integer); external;
  procedure wait; external;
  procedure delmode; external;
  procedure outbuf;  external;
  procedure ini_gr;  external;
```

```
  procedure move(x, y: integer); external;
  procedure draw(x, y: integer); external;
  procedure ink(k: integer); external;
  procedure detach; external;
  procedure citeste;
    begin
      read(fisier, inreg);
      inreg.nr := 0
    end;
  procedure scoate(var ch: char);
    begin
      with inreg do
        begin
        ch := buf[nr];
        if nr >= lgbuf then
          citeste
        else
          nr := succ(nr);
        end
    end;

%include julia2.pas;

  procedure citfis(var k: integer);
    var
      ch: char;
    begin
      scoate(ch);
      k := ord(ch)
    end;
%include julia3.pas;
  procedure citnumefis;
    begin
      write('Numele fisierului :');
      readln(numefis);
    end;

  begin { main }
    inibuf;
    citnumefis;
    reset(fisier, numefis);
    delmode;
    outbuf;
    citeste;
    deconverteste;
    scrie_datele;
    deseneaza_figura
  end.
```

Programul CM.pas

```
{[l-]}
{$NOCHECK}
program CM;
{ desenarea pe terminalul color a multimilor
  MANDELBROT folosind fisiere intermediare }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 8;
    us = 31;
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimq, x, deltap, deltaq, p, q, pmin, pmax, qmin,
    qmax, y, x1, x2, y2, r: real;
    opt, dr, sscp1, sscq1, scp1, scp2, scq1, scq2, inalt,
    lung, np, nq, k, kvechi, pvechi, qvechi: integer;
    ch: char;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure inibuf; external;
  procedure outchr(k: integer); external;
  procedure wait; external;
  procedure delmode; external;
  procedure outbuf; external;
  procedure ini_gr; external;
  procedure move(x, y: integer); external;
  procedure draw(x, y: integer); external;
  procedure ink(k: integer); external;
  procedure detach; external;
  procedure citeste;
    begin
      read(fisier, inreg);
        inreg.nr := 0
    end;
  procedure scoate(var ch: char);
```

```
  begin
    with inreg do
      begin
      ch := buf[nr];
      if nr >= lgbuf then
        citeste
      else
        nr := succ(nr);
      end
  end;
%include mandelb2.pas;
%include citfis.pas;
%include mandelb3.pas;

procedure scrie_datele;
  begin
    move(0, 24 + inaltime_ecran);
    outchr(us);
    outbuf;
    wait;
    writeln(' LG=', lung: 3, ' IN=', inalt: 3, ' PMI=',
            pmin: 3: 1, ' PMA=', pmax: 3: 1, ' QMI=',
            qmin: 3: 1, ' QMA=', qmax: 3: 1);
  end;
procedure citnumefis;
  begin
    write('Numele fisierului :');
    readln(numefis);
  end;
procedure pozc(var x, y: integer;
               var ch: char); external;
procedure afla_coord(var p, q: real;
                     var ch: char);
  begin
    pozc(np, nq, ch);
    p := (np - sscpl) * deltap + pmin;
    q := (nq - sscql) * deltaq + qmin;
  end;


begin { main }
  inibuf;
  citnumefis;
  reset(fisier, numefis);
  delmode;
  outbuf;
  citeste;
  deconverteste;
  scrie_datele;
  deseneaza_figura;
  repeat;
    afla_coord(p, q, ch);
```

```
    writeln('P=', p: 8: 6, 'Q=', q: 8: 6)
  until ord(ch) = 13;
end.
```

Programul DCJ.pas

```
{[l-]}
{$nocheck}
program DCJ;
{ desenarea pe terminalul alb-negru a multimilor
  JULIA folosind fisiere intermediare  }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 2;
    us = 31;
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimx, minimy, x, deltax, deltay, p, q, xmin, xmax,
    ymin, ymax, y, x1, x2, y2, r: real;
    opt, dr, sscx1, sscy1, scx1, scx2, scy1, scy2, inalt, lung,
    nx, ny, k, kvechi, xvechi, yvechi: integer;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure inibuf; external;
  procedure outchr(k: integer; external;
  procedure wait; external;
  procedure delmode; external;
  procedure outbuf; external;
  procedure ini_gr; external;
  procedure move(x, y: integer); external;
  procedure draw(x, y: integer); external;
  procedure ink(k: integer); external;
  procedure detach; external;
  procedure citeste;
    begin
      read(fisier, inreg);
      inreg.nr := 0
```

```
    end;
  procedure scoate(var ch: char);
    begin
      with inreg do
        begin
        ch := buf[nr];
        if nr >= lgbuf then
          citeste
        else
          nr := succ(nr);
        end
    end;
 %include julia2.pas;

 procedure citfis(var k: integer);
    var
      ch: char;
    begin
      scoate(ch);
      k := ord(ch) mod nrculori;
    end;


%include julia3.pas;

  procedure citnumefis;
    begin
      write('Numele fisierului :');
      readln(numefis);
    end;
  procedure pauza(tim: real);
    var
      x: real;
    begin
      x := time;
      while time < x + tim / 3600 do
    end;


  begin { main }
    inibuf;
    citnumefis;
    reset(fisier, numefis);
    ini_gr;
    outbuf;
    citeste;
    deconverteste;
    deseneaza_figura;
    pauza(10);
  end.
```

Programul DCM.pas

```
{[l-]}
{$NOCHECK}
program DCM;
{ desenarea pe terminalul alb-negru a multimilor
  MANDELBROT folosind fisiere intermediare  }
  const
    inaltime_ecran = 287;
    m = 100.0;
    kmare = 160;
    nrculori = 2;
    us = 31;
    lungime_ecran = 447;
    lgbuf = 509;
  type
    buffer = packed array [0..lgbuf] of char;
    bloc =
      record
        nr: integer;
        buf: buffer
      end;
    numefisier = packed array [1..12] of char;
    culori = array [0..7] of 0..7;
  const
    redef = culori(7, 3, 2, 6, 5, 1, 4, 0);
  var
    minimq, x, deltap, deltaq, p, q, pmin, pmax, qmin,
    qmax, y, x1, x2, y2, r: real;
    opt, dr, sscp1, sscq1, scp1, scp2, scq1, scq2, inalt, lung,
    np, nq, k, kvechi, pvechi, qvechi: integer;
    ch: char;
    fisier: file of bloc;
    inreg: bloc;
    numefis: numefisier;
  procedure inibuf; external;
  procedure outchr(k: integer); external;
  procedure wait; external;
  procedure delmode; external;
  procedure outbuf; external;
  procedure ini_gr; external;
  procedure move(x, y: integer); external;
  procedure draw(x, y: integer); external;
  procedure ink(k: integer); external;
  procedure detach; external;
  procedure citeste;
    begin
      read(fisier, inreg);
      inreg.nr := 0
    end;
  procedure scoate(var ch: char);
```

```
    begin
      with inreg do
        begin
        ch := buf[nr];
        if nr >= lgbuf then
          citeste
        else
          nr := succ(nr);
        end
    end;

%include mandelb2.pas;
%include citfis.pas;
%include mandelb3.pas;

  procedure scrie_datele;
    begin
      move(0, inaltime_ecran);
      outchr(us);
      outbuf;
      wait;
      writeln(' LG=', lung: 3, ' IN=', inalt: 3, ' PMI=', pmin:
              3: 1, ' PMA=', pmax: 3: 1, ' QMI=', qmin: 3: 1,
              ' QMA=', qmax: 3: 1);
    end;
  procedure citnumefis;
    begin
      write('Numele fisierului :');
      readln(numefis);
    end;
  procedure pozc(var x, y: integer;
                 var ch: char); external;
  procedure afla_coord(var p, q: real;
                       var ch: char);
    begin
      pozc(np, nq, ch);
      p := (np - sscpl) * deltap + pmin;
      q := (nq - sscql) * deltaq + qmin;
    end;
function getnr:char;external;


  begin { main }
    inibuf;
    citnumefis;
    reset(fisier, numefis);
    ini_gr;
    outbuf;
    citeste;
    deconverteste;
    scrie_datele;}
    deseneaza_figura;
   repear until getnr='Q';
```

```
    repeat;
      afla_coord(p, q, ch);
      writeln('P=', p: 8: 6, 'Q=', q: 8: 6)
    until ord(ch) = 13;
  end.
```

Proceduri citate cu `include`

JULIA1

```
procedure dreptunghi(x1, x2, y1, y2: real);
    var
      nx, ny: integer;
    begin
      scx1 := sscx1 + round((x1 - xmin) / deltax);
      scy1 := sscy1 + round((y1 - ymin) / deltay);
      dr := round((x2 - x1) / deltax);
      for ny := 0 to round((y2 - y1) / deltay) do
        for nx := 0 to dr do
          begin
          calcul(x1 + nx * deltax, y1 + ny * deltay, k);
          punct(nx, ny, k);
          end;
    end;
  procedure patrat(minimx, minimy: real);
    var
      nx, ny: integer;
    begin
      scx1 := sscx1 + round(( - minimx - xmin) / deltax);
      scx2 := scx1 + 2 * round(minimx / deltax);
      scy1 := sscy1 + round(( - minimy - ymin) / deltay);
      scy2 := scy1 + 2 * round(minimy / deltay);
      dr := round(minimx / deltax);
      for ny := 0 to round(2 * minimy / deltay) do
        for nx := 0 to dr do
          begin
          calcul(-minimx + nx * deltax,- minimy+ny * deltay,k);
          bipunct(nx, ny, k);
          end;
    end;
  procedure min(a, b, c, d: real;
              var minimx, minimy: real);
    begin
      minimx := a;
      if minimx > b then
        minimx := b;
      minimy := c;
      if minimy > d then
        minimy := d;
    end;
  procedure verifica_datele;
```

```
procedure schimba(var x, y: real);
  var
    z: real;
  begin
    z := x;
    x := y;
    y := z
  end;
begin
  if xmin > xmax then
    schimba(xmin, xmax);
  if ymin > ymax then
    schimba(ymin, ymax);
  if lung > lungime_ecran then
    lung := lungime_ecran;
  if inalt > inaltime_ecran then
    inalt := inaltime_ecran
end;
```

JULIA2

```
procedure deco_real(var x: real;
                    var nr: integer;
                    var f: buffer); external;
 procedure deco_int(var x: integer;
                    var nr: integer;
                    var f: buffer);  external;
 procedure deconverteste;
   begin {DECONVERTESTE}
     with inreg do
       begin
       deco_real(p, nr, buf);
       deco_real(q, nr, buf);
       deco_int(lung, nr, buf);
       deco_int(inalt, nr, buf);
       deco_real(xmin, nr, buf);
       deco_real(xmax, nr, buf);
       deco_real(ymin, nr, buf);
       deco_real(ymax, nr, buf);
       end
   end;
 procedure aprinde(x, y: integer);
   begin
     move(x, y);
     draw(x, y)
   end;
 procedure bipunct(nx, ny, k: integer);
   begin
     if nx = 0 then
       begin
       kvechi := k;
       xvechi := nx;
```

```
        yvechi := ny;
        end
      else if (k <> kvechi) or (nx = dr) then
        begin
        ink(redef[kvechi]);
        move(scx1 + xvechi,scy1 + yvechi);
        draw(scx1 + nx,scy1 + ny);
        move(scx2 - xvechi,scy2 - yvechi);
        draw(scx2 - nx,scy2 - ny);
        xvechi := nx;
        yvechi := ny;
        if (k <> kvechi) and (nx = dr) then
          begin
          ink(redef[k]);
          aprinde(scx1 + nx,scy1 + ny);
          aprinde(scx2 - nx,scy2 - ny);
          end;
        kvechi := k;
        end;
    end;
  procedure punct(nx, ny, k: integer);
    begin
      if nx = 0 then
        begin
        kvechi := k;
        ink(redef[k]);
        move( scx1, scy1 + ny);
        end
      else if (k <> kvechi) or (nx = dr) then
        begin
        ink(redef[kvechi]);
        draw(scx1 + nx,scy1 + ny);
        if (k <> kvechi) and (nx = dr) then
          begin
          ink(redef[k]);
          aprinde(scx1 + nx,scy1 + ny);
          end;
        kvechi := k;
        end;
    end;
```

JULIA3

```
procedure dreptunghi(x1, x2, y1, y2: real);
  var
    nx, ny: integer;
  begin
    scx1 := sscx1 + round((x1 - xmin) / deltax);
    scy1 := sscy1 + round((y1 - ymin) / deltay);
    dr := round((x2 - x1) / deltax);
    for ny := 0 to round((y2 - y1) / deltay) do
      for nx := 0 to dr do
```

```
        begin
        citfis(k);
        punct(nx, ny, k);
        end;
  end;
procedure patrat(minimx, minimy: real);
  var
    nx, ny: integer;
  begin
    scx1 := sscx1 + round(( - minimx - xmin) / deltax);
    scx2 := scx1 + 2 * round(minimx / deltax);
    scy1 := sscy1 + round(( - minimy - ymin) / deltay);
    scy2 := scy1 + 2 * round(minimy / deltay);
    for ny := 0 to round(2 * minimy / deltay) do
      begin;
      dr := round(minimx / deltax);
      for nx := 0 to dr do
        begin
        citfis(k);
        bipunct(nx, ny, k);
        end;
      end;
  end;
procedure min(a, b, c, d: real;
              var minimx, minimy: real);
  begin
    minimx := a;
    if minimx > b then
      minimx := b;
    minimy := c;
    if minimy > d then
      minimy := d;
  end;
procedure scalare;
  begin
    sscx1 := lungime_ecran div 2 + round(xmin / deltax);
    if sscx1 < 0 then
      sscx1 := 0
    else if sscx1 > lungime_ecran - lung then
      sscx1 := lungime_ecran - lung;
    sscy1 := inaltime_ecran div 2 + round(ymin / deltay);
    if sscy1 < 0 then
      sscy1 := 0
    else if sscy1 > inaltime_ecran - inalt then
      sscy1 := inaltime_ecran - inalt;
  end; {scalare}
procedure verifica_datele;
  procedure schimba(var x, y: real);
    var
      z: real;
    begin
      z := x;
      x := y;
```

```
      y := z
    end;
  begin
    if xmin > xmax then
      schimba(xmin, xmax);
    if ymin > ymax then
      schimba(ymin, ymax);
    if lung > lungime_ecran then
      lung := lungime_ecran;
    if inalt > inaltime_ecran then
      inalt := inaltime_ecran
  end;
procedure deseneaza_figura;
  begin
    verifica_datele;
    deltax := (xmax - xmin) / lung;
    deltay := (ymax - ymin) / inalt;
    scalare;
    min( - xmin, xmax, - ymin, ymax, minimx, minimy);
    if (minimx <= 0) or (minimy <= 0) then
      dreptunghi(xmin, xmax, ymin, ymax)
    else
      begin
      patrat(minimx, minimy);
      dreptunghi(xmin, minimx, ymin, - minimy);
      dreptunghi(xmin, - minimx, - minimy, ymax);
      dreptunghi( - minimx, xmax, minimy, ymax);
      dreptunghi(minimx, xmax, ymin, minimy);
      end;
    outbuf;
  end;
procedure scrie_datele;
  begin
    move(0, inaltime_ecran);
    outchr(us);
    outbuf;
    wait;
    writeln('P=', p: 5: 3, ' Q=', q: 5: 3, ' LG=', lung: 3,
            ' IN=', inalt: 3, ' XMI=', xmin: 3: 1, ' XMA=',
            xmax: 3: 1,' YMI=',ymin:3:1, ' YMA=', ymax: 3: 1);
  end;
```

MANDELB1

```
procedure calcul(p, q: real;
                 var k: integer);
  begin
    k := 0;
    x := 0;
    y := 0;
    repeat
      k := succ(k);
```

```
      x2 := sqr(x);
      y2 := sqr(y);
      x1 := x2 - y2 + p;
      y := 2 * x * y + q;
      x := x1;
      r := x2 + y2;
    until (r > m) or (k >= kmare);
    k := k mod nrculori;
  end;
procedure dreptunghi(p1, p2, q1, q2: real);
  var
    np, nq: integer;
  begin
    scp1 := sscp1 + round((p1 - pmin) / deltap);
    scq1 := sscq1 + round((q1 - qmin) / deltaq);
    dr := round((p2 - p1) / deltap);
    for nq := 0 to round((q2 - q1) / deltaq) do
      for np := 0 to dr do
        begin
        calcul(p1 + np * deltap, q1 + nq * deltaq, k);
        pune(chr(k))
        end;
  end;
procedure verifica_datele;
  procedure schimba(var x, y: real);
    var
      z: real;
    begin
      z := x;
      x := y;
      y := z
    end;
  begin
    if pmin > pmax then
      schimba(pmin, pmax);
    if qmin > qmax then
      schimba(qmin, qmax);
    if lung > lungime_ecran then
      lung := lungime_ecran;
    if inalt > inaltime_ecran then
      inalt := inaltime_ecran
  end;

procedure patrat(p1, p2, minimq: real);
  var
    np, nq: integer;
  begin
    scp1 := sscp1 + round((p1 - pmin) / deltap);
    scq1 := sscq1 + round(( - minimq - qmin) / deltaq);
    scq2 := scq1 + 2 * round(minimq / deltaq);
    dr := round((p2 - p1) / deltap);
    for nq := 0 to round(minimq / deltaq) do
```

```
      for np := 0 to dr do
        begin
        calcul(pl + np * deltap, - minimq + nq * deltaq, k);
        pune(chr(k));
        end;
   end;
 procedure min(a, b: real;
           var c: real);
   begin
     c := a;
     if c > b then
       c := b
   end;
```

MANDELB2

```
procedure deco_real(var x: real;
                    var nr: integer;
                    var f: buffer); external;
 procedure deco_int(var x: integer;
                    var nr: integer;
                    var f: buffer); external;
 procedure deconverteste;
   begin {deconverteste}
     with inreg do
       begin
       deco_int(lung, nr, buf);
       deco_int(inalt, nr, buf);
       deco_real(pmin, nr, buf);
       deco_real(pmax, nr, buf);
       deco_real(qmin, nr, buf);
       deco_real(qmax, nr, buf);
       end
   end;
 procedure aprinde(x, y: integer);
   begin
     move(x, y);
     draw(x, y)
   end;
 procedure punct(np, nq, k: integer);
   begin
     if np = 0 then
       begin
       kvechi := k;
       move(scpl, scql + nq);
       end
     else if (k <> kvechi) or (np = dr) then
       begin
       ink(redef[kvechi]);
       draw(scpl + np, scql + nq);
       if (k <> kvechi) and (np = dr) then
         begin
```

```
        ink(redef[k]);
        aprinde(scpl + np, scql + nq);
        end;
      kvechi := k;
      end;
    end;
```

MANDELB3

```
procedure dreptunghi(x1, x2, y1, y2: real);
    var
      np, nq: integer;
    begin
      scpl := sscpl + round((x1 - pmin) / deltap);
      scql := sscql + round((y1 - qmin) / deltaq);
      dr := round((x2 - x1) / deltap);
      for nq := 0 to round((y2 - y1) / deltaq) do
        for np := 0 to dr do
          begin
          citfis(k);
          punct(np, nq, k);
          end;
    end;
  procedure bipunct(np, nq, k: integer);
    begin
      if np = 0 then
        begin
        kvechi := k;
        pvechi := np;
        qvechi := nq;
        end
      else if (k <> kvechi) or (np = dr) then
        begin
        ink(redef[kvechi]);
        move(scpl + pvechi, scql + qvechi);
        draw(scpl + np, scql + nq);
        move(scpl + pvechi, scq2 - qvechi);
        draw(scpl + np, scq2 - nq);
        pvechi := np;
        qvechi := nq;
        if (k <> kvechi) and (np = dr) then
          begin
          ink(redef[k]);
          aprinde(scpl + np, scql + nq);
          aprinde(scpl + np, scq2 - nq);
          end;
        kvechi := k;
        end;
    end;
  procedure min(a, b: real;
                var c: real);
    begin
```

```
      c := a;
      if c > b then
        c := b
    end;
  procedure patrat(p1, p2, minimq: real);
    var
      np, nq: integer;
    begin
      scp1 := sscp1 + round((p1 - pmin) / deltap);
      scq1 := sscq1 + round(( - minimq - qmin) / deltaq);
      scq2 := scq1 + 2 * round(minimq / deltaq);
      dr := round((p2 - p1) / deltap);
      for nq := 0 to round(minimq / deltaq) do
        for np := 0 to dr do
          begin
          citfis(k);
          bipunct(np, nq, k);
          end;
    end;
  procedure scalare;
    begin
      sscp1 := lungime_ecran div 2 + round(pmin / deltap);
      if sscp1 < 0 then
        sscp1 := 0
      else if sscp1 > lungime_ecran - lung then
        sscp1 := lungime_ecran - lung;
      sscq1 := inaltime_ecran div 2 + round(qmin / deltaq);
      if sscq1 < 0 then
        sscq1 := 0
      else if sscq1 > inaltime_ecran - inalt then
        sscq1 := inaltime_ecran - inalt;
    end; {scalare}
  procedure verifica_datele;
    procedure schimba(var x, y: real);
      var
        z: real;
      begin
        z := x;
        x := y;
        y := z
      end;
    begin
      if pmin > pmax then
        schimba(pmin, pmax);
      if qmin > qmax then
        schimba(qmin, qmax);
      if lung > lungime_ecran then
        lung := lungime_ecran;
      if inalt > inaltime_ecran then
        inalt := inaltime_ecran
    end;
  procedure deseneaza_figura;
    begin
```

```
    verifica_datele;
    deltap := (pmax - pmin) / lung;
    deltaq := (qmax - qmin) / inalt;
    scalare;
    min( - qmin, qmax, minimq);
    if minimq <= 0 then
      dreptunghi(pmin, pmax, qmin, qmax)
    else
      begin
      patrat(pmin, pmax, minimq);
      dreptunghi(pmin, pmax, qmin, - minimq);
      dreptunghi(pmin, pmax, minimq, qmax);
      end;
    outbuf;
  end;
```

5.7.4. Biblioteca grafică GRAF.olb

Ţinînd seama că limbajul PASCAL nu are implementată posibilitatea utilizării zonelor de date comune (instrucţiunea COMMON din FORTRAN 77) biblioteca a fost astfel elaborată încît să fie eliminat acest obstacol, făcînd-se desigur apel la un artificiu impus de realitatea limbajului. În continuare, vom prezenta modul de construire a bibliotecii grafice, dar mai înainte prezentăm lista procedurilor conţinute şi care pot fi apelate din programe aplicative de grafică.

Proceduri grafice din biblioteca GRAF

procedure DRAW (x,y: integer)	trasează un segment de dreaptă din punctul curent pînă în punctul de coordonate absolute (x,y)
procedure MOVE (x,y: integer)	intră în modul grafic şi punctul curent devine punctul de coordonate absolute (x,y)
procedure APRINDE (x,y: integer)	aprinde punctul de coordonate absolute (x,y)
procedure TYPE__AT (x,y: integer; var line: txt; line_lg: integer)	tipăreşte tabloul LINE de tip TXT de max. 6 caractere şi de lungime LINE__LG începînd de la punctul de coordonate (x,y)
procedure COMENZI__SPECIALE	intră în modul de comenzi speciale
procedure INK (col:integer)	culoarea cernelii (pe terminalul color) devine COL ce poate fi din subdomeniul 0..7, iar ce s-a aflat pe ecran pînă la acel moment nu se schimbă

procedure PAPER (col: integer)	culoarea fondului pe care se scriu caracterele devine COL ce poate fi din subdomeniul 0..7
procedure SUPERPOSE	tipărirea caracterelor se face prin suprapunere
procedure DELMODE	tipărirea caracterelor se face prin ştergere
procedure TEKTRONIX__MODE	terminalul trece în mod TEKTRONIX
procedure CLEAR__SCREEN	şterge ecranul, iar culoarea fondului este culoarea setată prin PAPER
procedure INI__GR	iniţializează modul grafic, punctul curent devine (0,0), fondul VERDE, cerneala GALBEN şi se şterge ecranul
procedure PAPINK (i,j: integer)	anolog ca şi INI_GR, dar fondul are codul i, iar cerneala codul j
procedure STERGERE	ştergerea ecranului pentru DAF-2020
procedure VECTOR (vect: MOD__V)	pentru DAF-2020, trasarea vectorilor se face în mod uzual dacă MOD_V este '??' şi se realizează ştergerea dacă MOD__V este '@@'
procedure VECTOR	afectează rezultatul procedurii DRAW
procedure INTRA__PROG	pentru DAF-2020 se începe programarea caracterelor
procedure IESE__PROG	pentru DAF-2020 se termină programarea caracterelor
procedure COPIE	copierea ecranului terminalului DAF-2020 la imprimanta matricială
procedure INTROD	afişează cursorul cruce pe DAF-2020

Proceduri utilizate special pentru pachetul FRACTALI

procedure INIBUF	iniţializarea bufferului ce trebuie să se realizeze înaintea apelării oricărei proceduri ce foloseşte bufferul
procedure OUTBUF	transmite bufferul la terminal
procedure OUTCHR (n: integer)	depune în buffer caracterul CHR(n)

procedure OUTCAR (ch: char)	depune în buffer caracterul CH
procedure OUTBNR (nr, basis: integer)	scrie în buffer numărul NR în baza BASIS
procedure WAIT	forţează task-ul să aştepte terminarea operaţiilor de I/O; în mod obişnuit, cînd transmite bufferul la terminal, task-ul transmite comanda de tipărire a şirului de caractere şi continuă mai departe, fără să aştepte sfîrşitul operaţiei de tipărire
procedure DMOVE (lin, col: integer)	pentru terminalele alb-negru se poziţionează în linia LIN şi coloana COL, unde LIN ia valori în subdomeniul 1..24, iar COL ia valori în subdomeniul 1..80
procedure VID__INV	trecerea în modul video-invers
procedure VID__NOR	trecerea în modul video-normal
procedure CLEAR	ştergerea ecranului
procedure TIP__LA (lin, col: integer)	pentru TETRIS
procedure SETGR (term: TIP_TERMINAL)	setează parametrii folosiţi de DMOVE, VID__INV, VID_NOR şi CLEAR
procedure VT100	modul de lucru VT100 pentru terminal
procedure VT52	modul de lucru VT52 pentru terminal
procedure ANSI	modul de lucru ANSI pentru terminal
function ASKTERM	interoghează terminalul şi întoarce tipul terminalului; se ataşează terminalul astfel ca să nu acţioneze nici măcar tasta <CTRL/C>

Observaţie: *Biblioteca GRAF.olb conţine cîteva rutine (scrise în limbajul MACRO) care eclipsează unele puncte de intrare din PASLIB.olb în scopul utilizării aceluiaşi buffer de ieşire la rutinele de grafică şi la cele apelate de instrucţiunile de ieşire (WRITE, WRITELN, etc). Acest artificiu a fost impus de implementarea compilatorului PASCAL V02.H sub sistemul de operare RSX-11M.*

Proceduri sursă

TEKDR.pas

```
Program Tekdr;
{$nomain}
{$own}
{ modul de lucru TEKTRONIX  }
 Const
   white = 0;
   green = 2;
   yellow = 4;
   black = 7;
   max_len = 80;

   %Include lb:[1,1]graf.def
   %include termgr.def

 Var
   lyh, lyl, lxh, lxl: integer;
   { current pixel coordinate codes }

 Procedure Draw;
   Var
     hiy, loy, hix: integer;
   Begin
     if x< 0 then
       x := 0;
     if y < 0 then
       y := 0;
     X := X * 2;
     Y := Y * 2;
     { compute coordinate codes }
     hiy := y div 32 or 32;
     loy := y mod 32 or 96;
     hix := x div 32 or 32;
     { output them ( only if needed )
     and save them as current pixel }
     if hiy <> lyh then
       begin
       outchr(hiy);
       lyh := hiy
       end;
     if hix <> lxh then
       begin
       outchr(loy);
       outchr(hix);
       lyl := loy;
       lxh := hix
       end
     else if loy <> lyl then
```

```
      begin
      outchr(loy);
      lyl := loy
      end;
    lxl := x mod 32 or 64;
    outchr(lxl)
  End;

Procedure Move;
  Begin
    outchr(GS); { do not trace next "draw" }
    draw(x, y)
  End;

Procedure Aprinde;
  begin
    move(x, y);
    outchr(lxl)
  end;

Procedure Type_at;
  Var
    i: integer;
  Begin
    move(x, y);
    outchr(US);{Tektronix alpha mode;the first character}
    for i := 1 to line_lg do { will be typed at x,y }
      outchr(ord(line[i]))
  End;

Function cvt(n: integer): integer;
 { one digit integer to ASCII conversion }
  Begin
    cvt := ord('0') + n
  End;
Procedure comenzi speciale;
  begin
    outchr(esc);
    outcar('X')
  end;

Procedure Ink;
  Begin
    comenzi_speciale;
    outchr(ord('C'));
    outchr(cvt(col))
  End;

Procedure Paper;
  Begin
    comenzi_speciale;
    outchr(ord('F'));
    outchr(cvt(col))
```

```
    End;

  Procedure Superpose;
    Begin
      comenzi_speciale;
      outchr(ord('2'))
    End;

  Procedure Delmode;
    Begin
      comenzi_speciale;
      outchr(ord('3'))
    End;

  Procedure Tektronix_mode;
    Begin
      outchr(ESC);
      outchr(ord('1'))
    End;

  Procedure Clear_screen;
    Begin
      outchr(ESC);
      outchr(FF)    End;

  Procedure Ini_gr;
    Begin
      OUTCHR(ESC);
      OUTCHR(ORD('<'));
      papink(green, yellow)
    end;

  Procedure PAPINK;
    Begin
      tektronix_mode;
      delmode;
      paper(I);
      ink(J);
      clrtek;
    End;

  Procedure clrtek;
    begin
      clear_screen;
      { set false coordinates
      to current pixel to assure
      an effective move }
      lyh := 33;
      lyl := 97;
      lxh := 33;
      lxl := 65;
      move(0, 500)
    end;
```

```
PROCEDURE VECTOR;
  BEGIN
    COMENZI_SPECIALE;
    OUTCHR(SOH);
    OUTCHR(ORD(VECT[1]));
    OUTCHR(ORD(VECT[2]))
  END;
procedure vt100;
  begin
    comenzi_speciale;
    outchr(ord('1'))
  end;
procedure vt52;
  begin
    outchr(esc);
    outbuf;
    wait;
    writeln('[?2l');
  end;
procedure ansi;
  begin
    outchr(esc);
    outchr(ord('<'))
  end;
procedure intra_prog;
  begin
    comenzi_speciale;
    outchr(ord('P'))
  end;
procedure iese_prog;
  begin
    comenzi_speciale;
    outchr(ord('A'))
  end;
procedure copie;
  begin
    outchr(esc);
    outchr(etb);
    outbuf
  end;
procedure introd;
  begin
    outchr(esc);
    outchr(sub);
    outbuf
  end;
```

TERMGR.pas

```
program termgr;
 {$nomain}
```

```
{$own[l-]}
{ modul de lucru VT 100 }
 const
   dc4 = 20;
   dc2 = 18;
   esc = 27;
   bell = 7;
   bs = 8;
   ff = 12;

 var
   cdl, cd2, cd3, cvl, cv2, cv3, cvi2, cvn2, clrl,
   clr2, clr3, clr4,
   term: integer;

   %include outti.def;
   %include ast.def;
   %include termgr.def;

 procedure exitst(n: integer ) ;external ;
 procedure dmove;
   begin
     outchr(cdl);
     outchr(cd2);
     if term <> 1 then
       begin
       outchr(31 + lin);if  term=3 then outchr(ord(';'));
       outchr(31 + col)
       end
     else
       begin
       outchr(31 + col);
       outchr(31 + lin)
       end;
     outchr(cd3)
   end;
 procedure vid_inv;
   begin
     outchr(cvl);
     outchr(cv2);
     outchr(cvi2);
     outchr(cv3)
   end;
 procedure vid_nor;
   begin
     outchr(cvl);
     outchr(cv2);
     outchr(cvn2);
     outchr(cv3)
   end;
 procedure clear;
   begin
     outchr(clrl);
```

```
    outchr(clr2);
    dmove(1, 1);
    viD_nor;
    outbuf
  end;
procedure tip_la;
  begin
    dmove(lin, col * 2 - 1);
    outchr(ord(' '));
    outchr(ORD(' '))
  end;
procedure setgrf;
  begin
    term :=terminal;
    case terminal of
      1:
        begin
        cd1 := 0;
        cd2 := dc2;
        cd3 := 0;
        cv1 := 0;
        cv2 := dc4;
        cv3 := 0;
        cvi2 := ord('H');
        cvn2 := ord('@');
        clr1 := esc;
        clr2 := ord('Z');
        clr3 := 0;
        clr4 := 0
        end;
      2:
        begin
        cd1 := esc;
        cd2 := ord('Y');
        cd3 := 0;
        cv1 := 0;
        cv2 := 0;
        cv3 := 0;
        cvi2 := ord('N') - 64;
        cvn2 := ord('O') - 64;
        clr1 := esc;
        clr2 := ord('E');
        clr3 := 0;
        clr4 := 0
        end;
      3:
        begin
        cd1 := esc;
        cd2 := ord('[');
        cd3 := ord('H');
        cv1 := esc;
        cv2 := ord('[');
        cvi2 := ord('7');
```

```
            cvn2 := ord('0');
            cv3 := ord('m');
            clr1 := esc;
            clr2 := ord('[');
            clr3 := ord('2');
            clr4 := ord('J');
            outchr(esc);
            outchr(ord('<'));{Trece in mod ANSI din mod VT52 }
            end;
         otherwise
            BEGIN
            writeln(terminal);
            EXITST(2)
            END
         end;
      end; {setgrf}
```

GRAF.def

```
const esc=27;us=31;cr=13;
lf=10;
ff=12;gs=29;enq=5;soh=1;sub=26;
etb=23;
dc4=20;dc2=18;bel=7;bs=8;
ALB=0;ROSU=1;VERDE=2;GALBEN=4;MOV=6;NEGRU=7;

%include lb:[1,1]outti.def
%include lb:[1,1]tekdr.def
```

Observaţie: *se include la începutul fiecărui program ce foloseşte procedurile grafice.*

TEKDR.def

```
Const
  txt_len=6;
Type
  txt = packed array [1..txt_len] of char;
  MOD_V=PACKED ARRAY[1..2]OF CHAR;
Procedure Draw(x,y: integer); External;
Procedure Move(x,y: integer); External;
Procedure aprinde(x,y: integer); External;
Procedure Type_at(x,y: integer;
                  var line: txt;
                  line_lg: integer); External;
Procedure Comenzi_speciale; External;
Procedure Ink(col: integer); External;
Procedure Paper(col: integer); External;
Procedure Superpose; External;
Procedure Delmode; External;
```

```
Procedure Tektronix_mode; External;
Procedure Clear_screen; External;
Procedure Ini_gr; External;
Procedure PAPINK(I, J: INTEGER); EXTERNAL;
procedure STERGERE;external;
Procedure VECTOR(VECT: MOD_V);EXTERNAL;
procedure intra_prog;external;
procedure iese_prog;external;
procedure copie;external;
procedure introd;external;
```

OUTTI.def

```
Procedure Inibuf; External;
Procedure Outbuf; External;
Procedure Outchr(n: integer); External;
Procedure OUTCAR(CH: char); External;
Procedure OUTBNR(NR, BASIS: integer); External;
Procedure WAIT; External;
```

TERMGR.def

```
type tip_terminal=(daf2010,vdt52,daf_ansi,daf_vt52,daf_color);
procedure dmove(lin, col: integer);external;
procedure viD_inv;external;
procedure viD_nor;external;
procedure clear;external;
procedure tip_la(lin, col: integer);external;
procedure setgrf(terminal:tip_terminal);external;
procedure vt100;external;
procedure vt52;external;
procedure ansi;external;
function askterm:tip_terminal;external;
```

OUTPUT.pas

```
procedure wrtlin;external; { writeln }
procedure $b36;external;
procedure $b36; begin wrtlin end;

procedure outasc;external; { write('hgf': 7) }
procedure $b32;external;
procedure $b32; begin outasc end;

procedure outcha;external; { write('i': 6) }
procedure $b20;external;
procedure $b20; begin outcha end;
```

```
procedure outr0;external;
procedure $putch;external;
procedure $putch; begin outr0 end;

procedure putfil;external;
procedure $b60;external;
procedure $b60; begin putfil end;
procedure stctrl(ch:char);external;
```

Observaţie: *bufferul este trimis la terminal dacă se apelează* OUTPUT *sau* WRITELN, *sau dacă s-a umplut mai mult de jumătate din zona alocată lui.*

OUTPUT.mac

```
.title  soutpt
  .ident  /030590/
  .mcall  qiow$,dir$,setf$s,wtse$s

qiowal: qiow$   io.wal,5,5,,,,<buf,,40>
buf:    .blkb   512.
  .even
poscrt: .word   buf
mijloc= buf + 300.

.macro  chkpos             ; daca s-a umplut jumatate
  cmp poscrt,#mijloc  ; din buffer , se apeleaza
  blo .+6             ; OUTBUF
  call    outbuf
.endm

.macro  chkr0                      ; analog
  cmp r0,#mijloc
  blo .+6
  call    outbuf
.endm

outcar::
outchr::
  movb    2(sp),@poscrt
  inc poscrt
  chkpos
  mov (sp)+,(sp)
  return

outbuf::
  cmp poscrt,#buf
  blos    1$
  mov poscrt,qiowal+q.iopl+2
  sub #buf,qiowal+q.iopl+2
  dir$    #qiowal
  mov #buf,poscrt
```

```
1$: return

inibuf::
   mov #buf,poscrt
   setf$s  #5
   return

wait:: wtse$s  #5
   return

outr0::                ;putch
   movb    r0,@poscrt
   inc poscrt
   chkr0
   return

outreg::
   sub #buf,r0
   ble 1$
   mov r0,qiowal+q.iopl+2
   dir$    #qiowal
1$: mov #buf,r0
   mov r0,poscrt
   return

stctrl::                                    ; seteaza caracterul de
   clr qiowal+q.iopl+4                      ; control( conventie
   movb    2(sp),qiowal+q.iopl+4            ; FORTRAN implicit
   mov (sp)+,(sp)                           ; este ' ' )
   return                                   ; sau 0(NULL),'$', '0'

outasc::               ;$b32
adrasc= 14
lgsir = 12
lgtot = 10
rtadr = 6
   mov r0,(sp)             ; scrie in buffer un sir de
   mov r1,-(sp)            ; caractere
   mov r2,-(sp)            ; se apeleaza de exemplu cu
   mov poscrt,r0           ; write('ABC')
   mov lgtot(sp),r1
   sub lgsir(sp),r1
   ble 2$
1$: movb    #' ,(r0)+
   sob r1,1$
2$: add lgsir(sp),r1
   mov adrasc(sp),r2
3$: movb    (r2)+,(r0)+
   dec r1
   bgt 3$
   chkr0
   mov r0,poscrt
   mov (sp)+,r2
```

```
     mov (sp)+,r1
     mov (sp)+,r0
     mov (sp),6(sp)
     add #6,sp
     return
$wlnrc::
wrtlin::          ;$b36    ; se apeleaza la WRITELN
     cmp poscrt,#buf
     bne 1$
     clr -(sp)
     call    outchr              ; apel OUTBUF
1$: jmp outbuf

$b22:: tst -(sp)
outcha::          ;$b20    ; scrie un caracter in buffer
ch     = 10
lgtot = 6
     mov r0,(sp)
     mov r1,-(sp)             ; se apeleaza de exemplu cu
     mov poscrt,r0            ; write('S')
     mov lgtot(sp),r1
1$: dec r1
     ble 2$
     movb    #' ,(r0)+
     br 1$
2$: movb    ch(sp),(r0)+
     chkr0
     mov r0,poscrt
     mov (sp)+,r1
     mov (sp)+,r0
     mov (sp)+,(sp)
     mov (sp)+,(sp)
     return

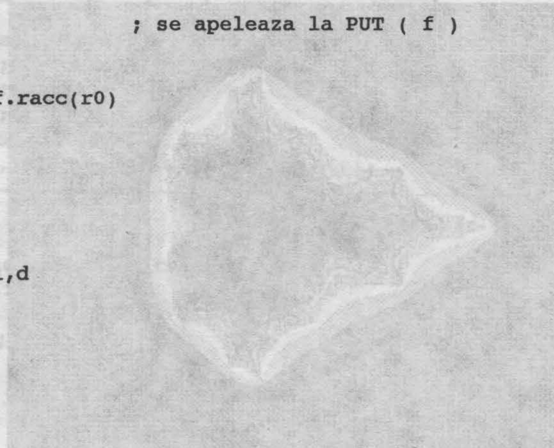
$b62::
     mov qiowal+q.iopl+4,-(sp)  ; se apeleaza la
     mov #'$,qiowal+q.iopl+4    ; BREAK( OUTPUT ) si
     call    outbuf ;break      ; forteaza scrierea cu
     mov (sp)+,qiowal+q.iopl+4  ; caracterul de control
     return                     ; '$'

putfil::          ; $b60
     jsr r5,$setio
     iot
     bit #10000,(r5)
     beq c664                   ; pentru fisiere de tip
     movb    @177776(r5),r0     ; f ⅄ file of INREG
     call    $putch
     br  c756
c664: mov 10(r5),r0
     beq c714
     bitb    #fd.ran,f.racc(r0)
```

```
    beq c714
    sub #1,f.rcnm+2(r0)
    sbc f.rcnm(r0)
c714:   mov 2(r5),r0
    call    @10(r0)             ; se apeleaza la PUT ( f )
    mov 10(r5),r0
    beq c756
    bitb    #fd.ran,f.racc(r0)
    beq c756
    bic #100000,(r5)
    mov 2(r5),r0
    call    @6(r0)
c756:   return

.psect  $$iov,ovr,gbl,d
.=.+2
    .word   wrtlin
.=.+4
    .word   outr0

.end
```



5.8. Reprezentarea fractalilor în TURBO–PASCAL

Prezentăm în continuare un program de desenare a structurilor fractale în varianta TURBO–PASCAL sub MS–DOS. Considerațiile teoretice privind teoria fractalilor au fost prezentate deja, astfel că nu mai insistăm decît asupra explicațiilor necesare obținerii unui desen folosind această sursă.

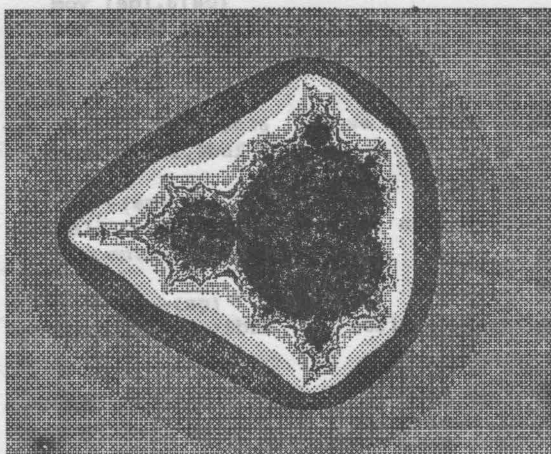
Utilizatorul va introduce fereastra în care „se vede" desenul în forma colț stînga–jos, colț dreapta–sus; apoi va introduce inițiala ('J' sau 'M') a structurii dorite. Pentru structura Julia, se mai introduc coordonatele constantei complexe c=(p,q).

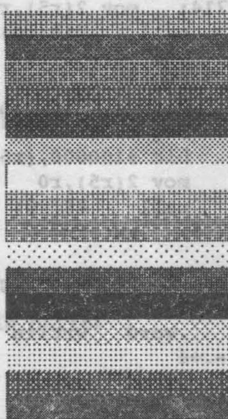
Apăsarea oricărei taste în timpul desenării (sau după terminarea ei) produce ieșirea în sistemul de operare MS–DOS. Pentru comparații ale culorilor asociate timpilor de evadare cu cele reprezentate pe ecran, s–a desenat în dreapta figurii paleta celor 16 culori utilizate.

Programul FRACTALS

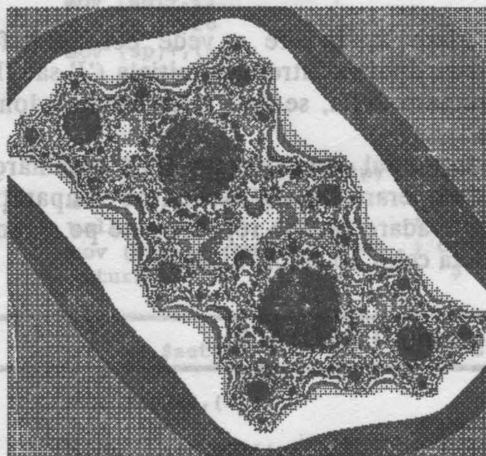
```
Program Fractals;
{
        Program de desenare a structurilor fractale
              Mandelbrot
              Julia

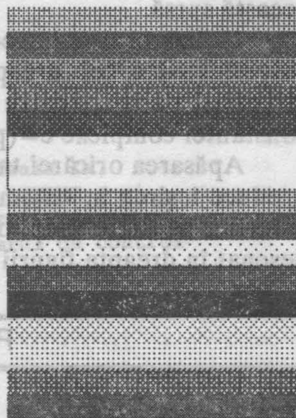
        Autor: Adrian Posea, 15 August 1991
}
```





Mulţimea MANDELBROT
(xmin, ymin, xmax, ymax) = (-2.5, -2, 1.7, 2)

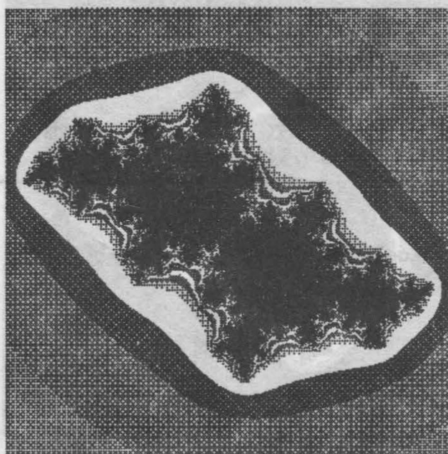
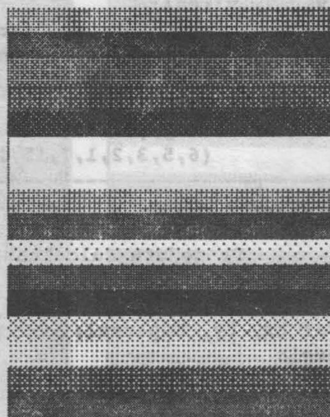




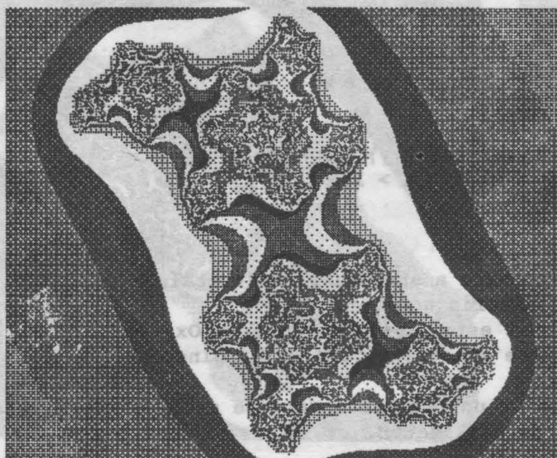
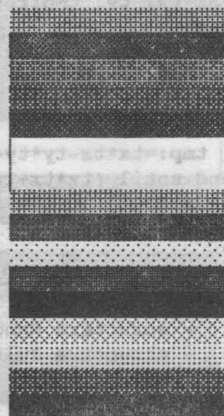
Mulţime JULIA
(xmin, ymin, xmax, ymax) = (-1.4, -1.2, 1.4, 1.2)
(p, q) = (-0.194, +0.6557)

```
---------------------------------------------------------------------
uses Graph, Crt;
 var x_min, y_min, x_max, y_max:real; centered:boolean;
 var x, y:real;
 var p, q:real; sx, sy, s:real;
 var i, j, k:integer; GraphMode:integer;
     h, w:integer; c:char;
 const l=64; M=100.0;
 colour: array [0..15] of 0..15 =
         (6,5,3,2,1,7,15,13,12,14,4,0,10,11,9,8);
---------------------------------------------------------------------
procedure Julia;
{
       Procedura calculeaza indicele k
       pentru iteratia Julia
}
var tx, ty, tmp:real;
begin
k:=0; tx:=x; ty:=y;
repeat
begin
  k:=k+1;
  tmp:=tx*tx-ty*ty+p; ty:=2*tx*ty+q; tx:=tmp;
end until (tx*tx+ty*ty > M) or (k > l);
end;
procedure Mandelbrot;
{
       Procedura calculeaza indicele k
       pentru iteratia Mandelbrot
}
var tx, ty, tmp:real;
begin
k:=0; tx:=x; ty:=y;
repeat
begin
  k:=k+1;
  tmp:=tx*tx-ty*ty+x; ty:=2*tx*ty+y; tx:=tmp;
end until (tx*tx+ty*ty > M) or (k > l);
end;
begin { main }
{
       Programul de desenare a structurilor fractale
       tine seama de simetria acestora:
   - structura Mandelbrot este simetrica fata de Ox;
   - structura Julia este simetrica fata de origine.

       Desenul se face pe un ecran standard pe care se reprezinta
  fereastra definita de utilizator.
}
  ClrScr; centered:=True;
  write('Init window: '); readln(x_min, y_min, x_max, y_max);
  if abs(x_max+x_min) > 0.001 then centered := False;
  if abs(y_max+y_min) > 0.001 then centered := False;
```

Mulţime JULIA
(xmin, ymin, xmax, ymax) = (-1.5, -1.5, 1.5, 1.5)
(p, q) = (-0.39054, +0.58679)

Mulţime JULIA
(xmin, ymin, xmax, ymax) = (-1.5, -1.2, 1.5, 1.2)
(p, q) = (-1.27334, +0.00742)

```
w:=640; sx:=(x_max-x_min)/w;
h:=350; sy:=(y_max-y_min)/h;
s:=sx; if sy > s then s:=sy;
h:=round((y_max-y_min)/s); if centered then h:=h div 2;
w:=round((x_max-x_min)/s);
repeat begin write('Julia or Mandelbrot ? ');c:=UpCase(ReadKey); end
until (c='J') or (c='M'); { Se alege intre Julia sau Mandelbrot
tastind initiala numelui structurii; alta litera este nepermisa.}
if c = 'J' then
 begin write('For Julia set enter p, q = ');
 read(p,q); end;
k:=Detect; InitGraph(k, GraphMode, 'f:\pascal\bgi');
x:=x_min;
for i:=0 to w do
 begin
  y:=y_min;
  for j:=0 to h do
  begin
  if KeyPressed then Halt; { Programul poate fi intrerupt
  in orice moment prin apasarea unei taste oarecare}
  if c='M' then
   begin
   Mandelbrot;
   if k < l then
     if centered then
     begin
      PutPixel(i, h+h-j, colour[k mod 16]);
      PutPixel(i, j, colour[k mod 16]);
     end else PutPixel(i, h-j, colour[k mod 16]);
   end
  else
   begin
   Julia;
   if k < l then
    if centered then
     begin
      PutPixel(i, h+h-j, colour[k mod 16]);
      PutPixel(w-i, j, colour[k mod 16]);
     end else PutPixel(i, h-j, colour[k mod 16]);
   end;
  y:=y+s;
  end;
 x:=x+s;
 end;
{ Desenare benzi colorate pentru comparatia culorilor
ce apar in structurile fractale reprezentate}
for i:=w+30 to 650 do
for j:=0 to 19 do
for k:=0 to 15 do
PutPixel(i, 20*k+j, colour[k]);
c:=ReadKey; {iesirea spre DOS se face apasind o tasta oarecare}
end.
```

5.9. Anexă

Elemente privind utilizarea terminalelor grafice DAF–2020, DAF–2020C, VDT–52S , ALFAGRAF–200

Terminalele grafice DAF–2020, DAF–2020C, VDT–52S, ALFAGRAF–200 sînt dispozitive de afișare alfanumerică și grafică avînd posibilități de lucru interactiv prin comunicație cu calculatorul electronic. Aceste dispozitive se utilizează într–o gamă largă de domenii: cercetare–proiectare asistată de calculator, aplicații tehnico–științifice, învățămînt, aplicații de evidență și gestiune, aplicații economice, medicină, geologie etc.

Terminalul DAF–2020

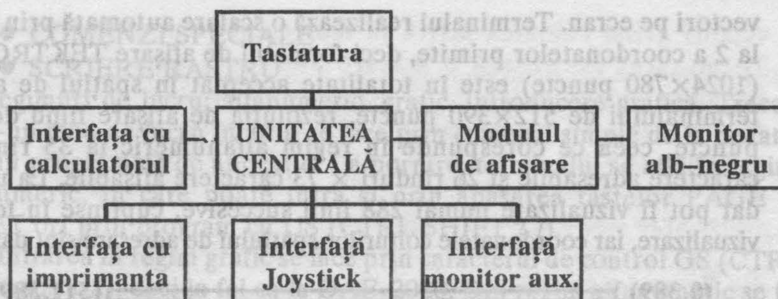
Terminalul are 3 moduri de lucru: TEKTRONIX 4010, VT 100, COMENZI SPECIALE și 4 regimuri de lucru: alfanumeric, grafic, introducere grafică, copie imprimantă. Trecerea dintr–un mod de lucru în altul se face prin comenzi simple de la tastatura terminalului sau din programele de aplicații. Modul de lucru în care se află terminalul la un moment dat este dat de indicatorul luminos CAPS, și anume , la pornire se află în mod TEKTRONIX 4010 (indicatorul CAPS este aprins permanent), dar în modul VT 100, la acționarea tastei CAPS indicatorul se stinge sau se aprinde.

Modul de lucru TEKTRONIX 4010 oferă utilizatorului posibilități grafice și hard copy, iar modul de lucru VT 100 oferă posibilități de editare alfanumerică și scroll. Îmbinarea celor două moduri de lucru oferă facilități sporite atît în dialogul cu calculatorul cît și pentru vizualizarea rezultatelor programelor de aplicații. Deoarece, la pornire, terminalul intră în modul TEKTRONIX 4010, trecerea în modul VT 100 se face prin trecerea în modul COMENZI SPECIALE cu tastele <CTRL> + <PF1> și apoi tasta 1, sau prin programul de aplicație cu <ESC> + <X> urmat de 1 de la comunicație. Revenirea în modul TEKTRONIX 4010 se face cu <ESC><1>, deci avem următoarea schemă:



Configurația generală a terminalului DAF–2020 este prezentată în figura ce urmează.

Unitatea centrală este formată dintr–un microcalcutator construit cu un microprocesor Z80, conține 2K memorie RAM, 10K memorie PROM



şi realizează toate funcţiile terminalului: scrie în memoria de ecran, generează vectori, gestionează interfeţele, etc. Modulul de afişare conţine o memorie de ecran de 24K (512×390 biţi–pixeli) şi generează semnalele de comandă necesare afişării pe ecran a informaţiei conţinută în memoria ecran.

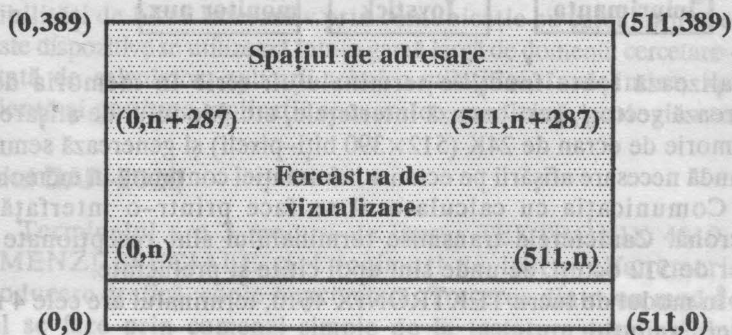
Comunicaţia cu calculatorul se face printr–o interfaţă serială asincronă. Caracterele transmise terminalului sînt recepţionate într–un buffer de 512 octeţi, de unde sînt apoi citite şi prelucrate.

În modul de lucru TEKTRONIX 4010, terminalul are cele 4 regimuri de lucru amintite.

În **regim alfanumeric**, numerotarea se face pentru linii de la 1 la 35 de sus în jos, iar pentru coloane de la 1 la 73 de la stînga la dreapta, poziţia iniţială (HOME) fiind linia 1 şi coloana 1. În acest regim există 2 moduri de deplasare a ferestei de vizualizare: urmărire automată şi poziţie fixă. În modul de urmărire automată fereastra de vizualizare se deplasează automat astfel încît cursorul alfanumeric să fie în permanenţă vizibil. În modul de poziţie fixă fereastra de vizualizare rămîne în poziţia în care se găsea la intrarea în acest mod. Fereastra se poate deplasa într–o altă poziţie fixă prin acţionarea tastelor săgeţi. În regim alfanumeric, la primirea unui cod ASCII între 20H şi 7EH se afişează pe ecran, într–o matrice de 7×11 puncte, caracterul corespunzător şi se mută cursorul în poziţia următoare. Ieşirea din regim alfanumeric, pentru trecerea în regim grafic, se face cu GS (CTRL]), pentru trecerea în regim introducere grafică, se face cu ESC SUB şi pentru trecerea în regim copie imprimantă, se face cu ESC ETB (CTRL W) după care se revine în regimul alfanumeric. În regim alfanumeric, terminalul este compatibil cu două standarde, şi anume ANSI şi VT52. Selectarea diferitelor facilităţi oferite de terminal se poate realiza printr–un mod de operare special numit SET–UP în care se poate intra prin apăsarea simultană a tastelor SCRL şi CTRL, acest mod numindu–se SET–UP A şi din care se poate trece în SET–UP B prin apăsarea tastei 5 din blocul alfanumeric.

În **regim grafic**, informaţia primită de la calculator sau tastatură este interpretată ca parametri cu ajutorul cărora se generează şi se afişează

vectori pe ecran. Terminalul realizează o scalare automată prin împărţirea la 2 a coordonatelor primite, deci formatul de afişare TEKTRONIX 4010 (1024×780 puncte) este în totalitate acceptat în spaţiul de adresare al terminalului de 512×390 puncte, rezoluţia de afişare fiind de 512×288 puncte, ceea ce corespunde în regim alfanumeric la 35 rînduri × 73 caractere adresabile şi 26 rînduri × 73 caractere afişabile. La un moment dat pot fi vizualizate numai 288 linii succesive, cuprinse în fereastra de vizualizare, iar coordonatele colţurilor spaţiului de adresare sînt date în figură.



Spaţiul de adresare poate fi vizualizat în întregime prin deplasarea ferestrei de vizualizare. Iniţial, fereastra de vizualizare cuprinde liniile 102–389 şi se poate deplasa cu ajutorul tastelor săgeţi care au numai efect local. În regim grafic, codurile ASCII care nu sînt coduri de control sînt folosite pentru completarea coordonatelor punctului grafic curent (X,Y). În momentul în care s–au completat coordonatele X,Y ale unui punct , se uneşte printr–un vector acel punct cu punctul ale cărui coordonate au fost completate anterior (punctul iniţial), noul punct devine punct iniţial urmînd ca el să fie unit cu altul în momentul în care se completează din nou coordonatele X,Y ale unui nou punct. Coordonatele X,Y au valori de la 0 la 1023 şi de aceea valorile lor sînt date prin 10 biţi.

În regimul de **introducere grafică**, apare pe ecran un cursor cruce ce poate fi deplasat în spaţiul de adresare al terminalului cu ajutorul tastelor săgeţi.

În regim **copie imprimantă**, terminalul trimite la imprimantă copia ecranului.

În modul de lucru COMENZI SPECIALE, se pot schimba diferiţi parametri, se pot activa facilităţi folosite în celelalte moduri sau se poate trece din modul TEKTRONIX în modul VT 100 cu ajutorul unor comenzi speciale. Intrarea în acest mod se face numai din modul TEKTRONIX, de la tastatură prin <CTRL> + <PF1> sau ESC X în programele de aplicaţie.

Terminalul DAF–2020C

Terminalul are monitorul color, 3 moduri de lucru:

- TEKTRONIX 4010

- COMENZI SPECIALE
- SCRIERE RASTRU

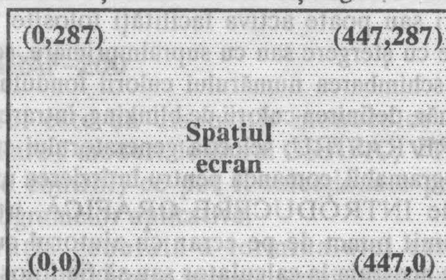
şi 3 regimuri de lucru: alfanumeric, grafic, introducere grafică. Trecerea dintr–un mod de lucru în altul se face prin comenzi simple de la tastatură sau din programele de aplicaţie. La pornire, terminalul se află în regimul alfanumeric, în care poate intra şi prin apasărea tastelor PAGE sau RESET ori prin program cu US (CTRL SHIFT ? /).

Intrarea în regim grafic se face prin caracterul de control GS (CTRL] de la tastatură), deci la fel ca la DAF–2020. Din regim alfanumeric se iese cu ESC SUB şi se trece în regim introducere grafică. În modul COMENZI SPECIALE, se intră cu CTRL PF1 de la tastatură sau cu ESC X prin program, deci ca şi la DAF–2020.

În comparaţie cu terminalul DAF–2020, terminalul DAF–2020C are o configuraţie generală asemanătoare, cu deosebirea că DAF–2020C nu are interfaţa pentru imprimantă şi pentru joystick.

Unitatea centrală este formată dintr–un microcalculator construit cu un microprocesor Z80 ce conţine 1K memorie RAM şi 10K memorie PROM. Modulul de afişare conţine o memorie de ecran de 64K, patru plane a cîte 448×288 biţi (pixeli). Facilităţile grafice şi alfanumerice oferite de DAF–2020C în modul TEKTRONIX 4010, şi folosirea simultană a 16 culori selectabile dintr–un număr de 64, permit utilizatorului un dialog mai eficient cu calculatorul prin afişarea unei cantităţi mari de informaţii. Comunicaţia cu calculatorul se face printr–o interfaţă serială asincronă. Caracterele transmise terminalului DAF–2020C sînt recepţionate într–un buffer de 512 octeţi, de unde sînt apoi citite şi prelucrate.

În modul de lucru TEKTRONIX 4010, terminalul are 3 regimuri de lucru: alfanumeric, grafic şi introducere grafică. În regim alfanumeric se transmit şi se afişează în culoarea de lucru (iniţial culoarea 4 = galben) caractere alfanumerice şi terminalul foloseşte modul de afişare pagină. Prin utilizarea tastei PAGE, indiferent de regimul de lucru, se revine la modul alfanumeric, cu ştergerea ecranului, iar prin utilizarea tastei RESET se obţine acelaşi efect dar fără ştergerea ecranului.



În regim grafic, informaţia primită de la calculator sau de la tastatură este interpretată ca parametri cu ajutorul cărora se generează şi se afişează

vectori avînd culoarea activă în momentul transmiterii coordonatelor. Terminalul realizează o scalare automată prin împărţirea cu 2 a coordonatelor primite. Pentru ecran spaţiul de adresare este de 488×288 puncte, ceea ce corespunde în regim alfanumeric la 24 rînduri şi 64 caractere pe rînd. În regim alfanumeric ecranul este împărţit în 24 rînduri şi 64 coloane, numerotarea făcîndu-se de sus în jos pentru linii şi de la stînga spre dreapta pentru coloane.

În regim grafic, coordonatele colţurilor spaţiului de adresare sînt cele din figura anterioară.

Punctele ecranului au asociat un număr de la 0 la 15 (0 la F în hexazecimal) ce precizează care din cele 16 culori afişabile sînt folosite pentru colorarea punctului respectiv. Aceste numere sînt :

0 = alb	1 = roşu	2 = verde	3 = albastru
4 = galben	5 = cian	6 = magenta	7 = negru
8 = 7	9 = 6	10 = 5	11 = 4
12 = 3	13 = 2	14 = 1	15 = 0

Pentru specificarea unei culori s-a ales metoda RGB ('red', 'green', 'blue') prin care se precizează procentele de roşu, verde şi albastru conţinute în culoarea respectivă. Pornind de la cele 16 culori, folosind 4 trepte de intensitate, se obţin cele 64 de culori.

Deoarece, în regim grafic spaţiul de lucru este 448×288 puncte şi coordonatele X şi Y iau valori de la 0 la 1023, unui punct (X,Y) i se asociază pe ecran un punct de coordonate (X/2, Y/2), punctul (0,0) fiind în colţul din stînga jos.

În regim alfanumeric, la primirea unui cod ASCII cuprins între 20H şi 7EH caracterul corespunzător se afişează pe ecran în culoarea de lucru selectată în poziţia curentă, într-o matrice de 7×12 puncte. În varianta standard cu 16 culori, codurilor ASCII li se pot asocia descrieri de 7×12 puncte care să înlocuiască definiţia standard, aceasta dacă s-a selectat generatorul prin intermediul modului de lucru COMENZI SPECIALE.

În modul de COMENZI SPECIALE utilizatorul poate schimba diferiţi parametri sau poate activa facilităţi folosite în celelalte moduri: trecerea în afişare cu ştergere sau cu supraimprimare, schimbarea numărului culorii de lucru, schimbarea numărului culorii fondului, schimbarea culorii asociate unui număr, definirea culorii de blinking, intrarea şi ieşirea din modul de lucru SCRIERE RASTRU, selecţia generatorului standard, completarea generatorului programabil, comanda pentru întîrzierea unei execuţii.

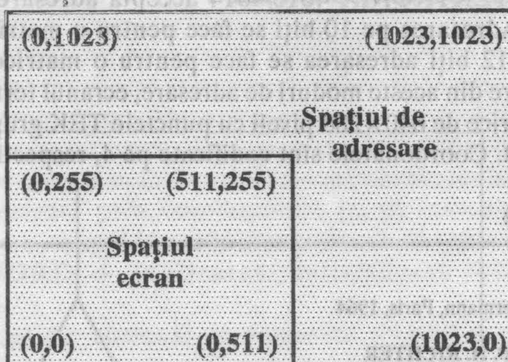
În regim de INTRODUCERE GRAFICĂ, utilizatorul poate să selecteze un anumit punct de pe ecran cu ajutorul cursorului cruce şi să transmită poziţia acestuia la calculator sau să fie folosită în programele de aplicaţie. Deplasarea cursorului cruce se realizează cu tastele săgeţi.

Terminalul VDT–52S

Terminalul este monocolor și are 4 regimuri de lucru: ALFANUMERIC (alpha mode), GRAFIC (vector mode), INTRODUCERE GRAFICĂ (gin mode — graphic input) și COPIE IMPRIMANTĂ.

În regimul alfanumeric, terminalul VDT–52S (Video Display Terminal 52S) este compatibil cu terminalele DEC VT52 și VDT 40C, iar în regim grafic și introducere grafică este compatibil TEKTRONIX 40xx. Unitatea centrală este formată dintr–un microcalculator construit cu microprocesorul Z80 și conține 1K memorie RAM și 6K memorie PROM.

În regim grafic utilizatorul are la dispoziție un spațiu grafic de 512×256 puncte (pixeli), iar spațiul de adresare este 1024×1024 puncte, coordonatele X și Y luînd valori de la 0 la 1023:



Intrarea în regim grafic se face la fel ca și la terminalele DAF–2020 și DAF–2020C și anume cu caracterul de control GS (CTRL] de la tastatură). Pentru selectarea parametrilor de lucru ai terminalului se utilizează tasta SET–UP care realizează afișarea sau modificarea stării cheilor corespunzătoare parametrilor, utilizînd tastele funcționale F1, F2, ..., F14. În regim de introducere grafică, terminalul se comportă analog cu terminalele DAF–2020 și DAF–2020C. În regim alfanumeric, este compatibil cu modul de lucru VT52.

Terminalul ALFAGRAF–200

Terminalul este un dispozitiv monocolor de afișare grafică și alfanumerică și are 5 moduri de lucru: GRAFIC, VT100, VT52, VT200 (cu 7 sau 8 biți de control) ce pot fi selecționate prin SET–UP sau prin intermediul programelor aplicative.

Configurația generală a terminalului este identică cu cea a terminalului DAF–2020, prin urmare are și interfețe pentru imprimantă și joystick. Unitatea centrală este formată dintr–un microcalculator construit cu un microprocesor Z80–CPU și conține 8K memorie RAM și 20k

memorie PROM. Modulul de afişare conţine o memorie de 23K, adică 640×288 biţi (puncte). În modul de lucru grafic, terminalul execută funcţii similare terminalelor compatibile TEKTRONIX 4010/4014. În modul de lucru VT52, se foloseşte compatibilitatea cu terminalul VT52 al firmei DEC. În modul de lucru VT100, se execută funcţii standard ANSI folosindu-se compatibilitatea strictă cu terminalul VT100 al firmei DEC. În modul de lucru VT200, se execută funcţii standard ANSI folosite pentru programe aplicative care necesită caractere de control de 7 biţi sau 8 biţi.

În regim grafic, vectorii sînt trasaţi între punctele cu coordonate absolute, ce reprezintă valori ale punctelor cu o distribuţie apropiată de pixelul fizic corespunzător. Caracterul de control GS permite trecerea din regim alfanumeric în regim grafic, definind astfel începutul desenării unui vector. Modul TEKTRONIX 4010/4014 acceptă adresarea fie pe 10 biţi, fie pe 12 biţi. Adresarea pe 10 biţi se face pentru o matrice de 1024×768 puncte, iar pe 12 biţi adresarea se face pentru o matrice de 4096×3072 puncte. În oricare din aceste moduri de adresare, ecranul terminalului rămîne definit de o matrice de 640×288 pixeli cu punctele TEK grupate lîngă pixelul cel mai apropiat. Coordonatele sînt codificate pe 4, respectiv pe 5 octeţi.

5.10. Bibliografie

[1] GASTON JULIA
OEUVRES, vol 1, Hermann, Paris, 1964

[2] H. O. PEITGEN, P. H. RICHTER
Beauty of Fractals, Images of Complex Dynamical Systems, Springer–Verlag, 1986

[3] B. MANDELBROT
The Fractal Geometry of Nature, Freeman, 1982

[4] R. L. DEVANEY, LINDA KEEN
Chaos and fractals, The mathematics behind the computer graphics, vol. 39, Proceedings of Simposia in Applied Mathematics, 1989

[5] UWE BECK
Computer Graphik, Bilder und Programme zu Fraktalen, Chaos und Selbstähnlichkeit, 1987

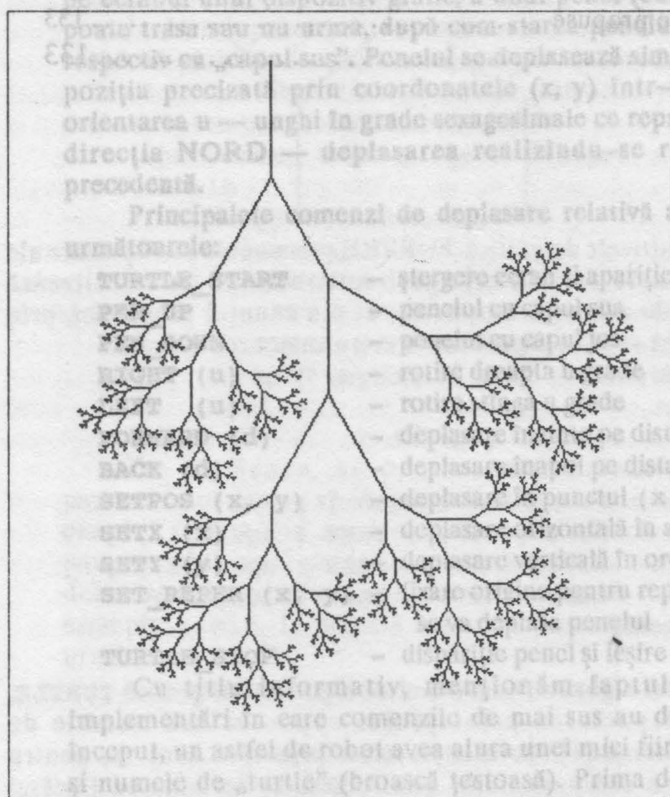
[6] * * * *DAF–2020C — Manual de prezentare şi utilizare*, FEPER, 1988

[7] * * * *DAF–2020 — Manual de prezentare şi utilizare*, FEPER, 1986

[8] * * * *ALFAGRAF–200 — Manual de prezentare şi utilizare*, FEPER, 1988

[9] * * * *VDT–52S — Manual de prezentare şi utilizare*, ICE, 1987

APLICAȚII ÎN GEOMETRIA „TURTLE”



6

6

6.1. Introducere

„TURTLE geometry” sau „TURTLE graphics” reprezintă un stil de grafică pe calculator introdus de limbajul de inteligență artificială LOGO și care are la bază un set de comenzi (operații) primitive pentru deplasarea pe ecranul unui dispozitiv grafic, a unui penel (cursor grafic) virtual care poate trasa sau nu urma, după cum starea penelului este cu „capul jos”, respectiv cu „capul sus”. Penelul se deplasează similar unui robot care are poziția precizată prin coordonatele (x, y) într–un sistem cartezian și orientarea u — unghi în grade sexagesimale ce reprezintă azimutul față de direcția NORD — deplasarea realizîndu–se relativ față de poziția precedentă.

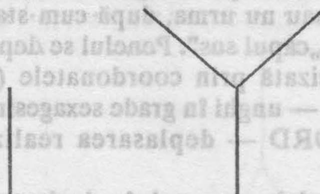
Principalele comenzi de deplasare relativă a penelului virtual sînt următoarele:

`TURTLE_START`	– ștergere ecran și apariție penel
`PEN_UP`	– penelul cu capul sus
`PEN_DOWN`	– penelul cu capul jos
`RIGHT (u)`	– rotire dreapta u grade
`LEFT (u)`	– rotire stînga u grade
`FORWARD (d)`	– deplasare înainte pe distanța d
`BACK (d)`	– deplasare înapoi pe distanța d
`SETPOS (x, y)`	– deplasare în punctul `(x,y)`
`SETX (x)`	– deplasare orizontală în abscisa x
`SETY (y)`	– deplasare verticală în ordonata y
`SET_REPER (x, y)`	– fixare origine pentru reperul de axe față de care se va deplasa penelul
`TURTLE_STOP`	– dispariție penel și ieșire din mod grafic

Cu titlu informativ, menționăm faptul că au fost realizate implementări în care comenzile de mai sus au dirijat un robot real. La început, un astfel de robot avea alura unei mici ființe cu carapace, de unde și numele de „turtle” (broască țestoasă). Prima denumire a fost utilizată de către neurofiziologul GREY WALTER pentru niște roboți electronici construiți și experimentați în Anglia la începutul anilor ’60.

Implementările pentru realizarea comenzilor care stau la baza geometriei generate de stilul „turtle", sînt dintre cele mai deosebite, ţinînd seama de performanţele obţinute în domeniul graficii computerizate. Un impact deosebit în acest sens l-a avut implementarea din limbajul inteligenţei artificiale **LOGO**. După cum vom vedea, principiul recursivităţii şi al transparenţei referenţiale, sînt principii de bază în grafica generată de geometria TURTLE.

Ca exemplu, privind utilizarea comenzilor de tip TURTLE, considerăm un algoritm pentru generarea (desenarea) unui model simplu de arbore binar stilizat. Presupunem că un arbore de vîrsta 0 este reprezentat de un punct, un arbore de vîrsta 1 este reprezentat de un segment de dreaptă şi, în general, un arbore de vîrsta n > 0 este reprezentat de un segment care se ramifică la vîrf în două ramuri care, abstracţie făcînd de orientare, sînt ambele izomorfe cu un arbore de vîrsta n-1. Pentru vîrstele 0, 1, 2 obţinem următoarele modele:



Utilizînd primitivele de grafică TURTLE (comenzile de deplasare ale penelului menţionate anterior), algoritmul recursiv care realizează (generează) astfel de desene se obţine direct din enunţul problemei, prin urmare, translatarea sa într-o procedură are următoarea formă:

```
procedure TREE ( age : integer ) ;
 begin
  if age = 0 then { point }
         else begin
            FORWARD ( length ) ;
            LEFT ( 45 ) ; TREE ( age - 1 ) ;
            RIGHT ( 90 ); TREE ( age - 1 ) ;
            LEFT ( 45 ) ; BACK ( length )
        end
 end;
```

Reuşita acestui algoritm, ca şi a multor algoritmi de grafică TURTLE, se bazează pe respectarea principiului cunoscut sub numele de „transparenţă referenţială". În esenţă, acest principiu cere ca, pentru îmbinarea corectă a componentelor unui desen, la terminarea fiecărei proceduri penelul să se găsească exact în aceeaşi poziţie (coordonate şi orientare) ca la începutul apelării procedurii. Prin inducţie, vom verifica

faptul că pentru algoritmul prezentat mai sus, este respectat principiul transparenței referențiale. Într-adevăr, pentru age=0 ipoteza este evident verificată. Pentru age=1, execuția urmează ramura else, deci penelul se deplasează pe direcția NORD, se rotește spre stînga cu 45°, se apelează TREE cu age=0, deci se desenează un punct, se rotește spre dreapta cu 90° și se apelează TREE cu age=0, deci se deseneză un punct peste cel anterior, în sfîrșit se rotește spre stînga cu 45° (adică s-a revenit la direcția NORD inițială) și se deplasează înapoi pe distanța pe care a parcurs-o la început, prin urmare penelul va avea direcția NORD și poziția sa va fi exact în punctul inițial, cînd a avut loc apelul TREE. Acum, presupunînd că ipoteza este verificată pentru cazul age=k, s-o verificăm pentru cazul age=k+1. Prin apelul TREE cu age=k+1, execuția urmează ramura else și, conform celor explicate anterior, schimbarea totală de orientare este egală cu o întoarcere la stînga cu 45+0-90+0+45 = 0°. Similar, deplasarea totală este egală cu length+0+0-length = 0. Prin urmare, în cazul procedurii TREE, este respectat principiul transparenței referențiale.

În forma procedurii de mai sus, length este o constantă sau o variabilă globală. În acest sens, putem să alegem o implementare mai rafinată, în care lungimile segmentelor (ramurilor) se micșorează cu vîrsta, deci vom avea următoarea formă:

```
procedure TREE ( age : integer ; length :real ) ;
begin
  if age = 0 then { point }
  else begin
    FORWARD ( length ) ;
    LEFT ( 45 ) ; TREE ( age-1, length/2 );
    RIGHT ( 90 ); TREE ( age-1, length/2 );
    LEFT ( 45 ) ; BACK ( length )
  end
end;
```

De asemenea, se poate considera varianta de eliminare a parametrului pentru vîrsta arborelui age, folosit numai pentru detectarea cazului limită, de oprire a recursiei. Dacă acceptăm eliminarea parametrului age, putem să acceptăm că nu are rost să desenăm și să dezvoltăm în continuare ramuri mai scurte decît o valoare dată, de exemplu limit. În această variantă, procedura poate fi concepută sub următoarea formă:

```
procedure TREE ( length : real ) ;
begin
  if length < limit then { point }
  else begin
    FORWARD ( length ) ;
    LEFT ( 45 ) ; TREE ( length/2 );
```

```
            RIGHT ( 90 ); TREE ( length/2 );
            LEFT ( 45 ) ; BACK ( length )
         end
      end
```

Precizăm faptul că se pot imagina și diferite alte variante, în care să intervină, de exemplu, și factori aleatori, astfel că se poate ajunge la modelarea și studierea diverselor fenomene organice sau la crearea de forme picturale cu ajutorul calculatorului.

În continuare, prezentăm un desen din categoria deosebit de interesantă a fractalilor. Nu vom prezenta detalii teoretice privind teoria fractalilor (unele dintre acestea au fost date în capitolul 5). Ne propunem să realizăm numai un algoritm pentru generarea unei familii de curbe închise (curbe KOCH) care să aproximeze din ce în ce mai bine conturul unui fulg de zapadă ideal. Acesta va fi un exemplu clasic de curbă de lungime infinită cuprinsă într–un disc mărginit și care conține graficul unei funcții continue, dar nederivabile. Primele curbe din familie sînt prezentate în fig. 5–1 din capitolul 5 unde se dau cîteva detalii despre generarea curbelor KOCH și unde se precizează că acestea sînt curbe fractale. Pentru un studiu și o cercetare mai amănunțită se recomandă revenirea la capitolul 5 și consultarea bibliografiei de la capitolele 5 și 6, ori consultarea unui cunoscător binevoitor al acestui domeniu.

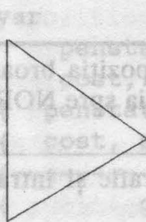
Prima curbă a șirului, curba de gradul (nivel) 0 este un triunghi echilateral de latură finită. În general, curba de gradul (nivel) n+1 se obține din curba de gradul n, înlocuind fiecare latură a acesteia printr–o linie poligonală formată din 4 segmente avînd ⅓ din lungimea laturii inițiale, astfel încît primul și ultimul segment coincid cu prima și ultima treime a laturii inițiale, iar celelalte două segmente, împreună cu treimea din mijloc, formează un triunghi echilateral cu vîrful spre exteriorul curbei. Cu alte cuvinte, scoatem treimea din mijloc și o înlocuim cu două segmente cu care aceasta ar forma un triunghi echilateral avînd vîrful spre exteriorul curbei.

Programarea recursivă a acestui algoritm este foarte naturală: avem nevoie de două proceduri, una recursivă pentru a desena o latură și alta nerecursivă pentru a închide curba. Procedurile SIDE, respectiv SNOW_FLAKE realizează acest lucru:

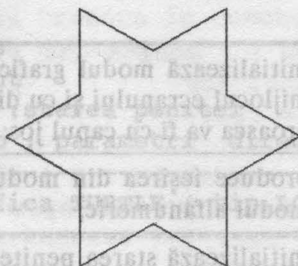
```
procedure SIDE ( n :integer ; length :real );
begin
  if n = 0 then FORWARD ( length )
  else begin
    SIDE ( n-1, length / 3 ) ; LEFT ( 60 ) ;
    SIDE ( n-1, length / 3 ) ; RIGHT ( 120 ) ;
    SIDE ( n-1, length / 3 ) ; LEFT ( 60 ) ;
    SIDE ( n-1, length / 3 )
```

```
      end
   end;

   procedure SNOW_FLAKE ( length : real );
   begin
      SIDE ( n, length ) ; RIGHT ( 120 ) ;
      SIDE ( n, length ) ; RIGHT ( 120 ) ;
      SIDE ( n, length ) ; RIGHT ( 120 )
   end;
```



n=0



n=1



n=2

Algoritmul prezentat prin cele două proceduri desenează un fulg de zapadă de ordinul n, parcurgerea făcîndu-se în sens invers trigonometric. Limita, după n, a acestui șir de curbe este un exemplu clasic de curbă de lungime infinită cuprinsă într-un disc mărginit. În cele ce urmează vom prezenta modul de implementare a comenzilor de tip TURTLE în limbajele PASCAL și C sub sistemul de operare RSX-11M pe minicalculatoarele din familiile CORAL sau INDEPENDENT. Vom folosi facilitățile oferite de bibliotecile grafice DAFPAS și DAFC prezentate în capitolele 1 și 2.

De asemenea, vom prezenta modul lor de utilizare în cadrul unor programe care generează diverse construcții geometrice (curbe fractale etc.).

Pe microcalculatoarele compatibile IBM-PC sub sistemele de operare PC-DOS și MS-DOS există unele implementări, de exemplu în TURBO PASCAL (versiunea 3.0 Copyright 1986, BORLAND INTERNATIONAL Inc.; pentru versiunile 5.0-6.0 vezi 6.2.1).

6.2. Grafica TURTLE în limbajul PASCAL

Pentru realizarea de programe de grafică TURTLE este nevoie de elaborarea unei baze de primitive grafice care să aibă funcțiile precizate mai sus. Indicațiile pe care le vom da în continuare, împreună cu informațiile existente în textul sursă al procedurilor grafice, credem că vor

fi suficiente pentru a înţelege modul lor de elaborare, precum şi funcţiile lor.

Vom folosi următoarele variabile globale:

```
PENSTATUS      : boolean ;
COST , SINT    : real ;
```

Variabila `PENSTATUS` indică starea peniţei (`true` pentru peniţă cu capul jos, şi `false` pentru peniţa cu capul sus), iar variabilele `COST`, `SINT` reprezintă valorile cos(u), sin (u), unde u este direcţia curentă a peniţei (broaştei) faţă de direcţia NORD; unghiul u se va măsura în grade sexagesimale. Procedurile grafice pentru grafica TURTLE vor fi următoarele:

procedure TURTLE_START;	iniţializează modul grafic şi poziţia broaştei în mijlocul ecranului şi cu direcţia spre NORD, iar broasca va fi cu capul jos;
procedure TURTLE_STOP;	produce ieşirea din modul grafic şi intrarea în modul alfanumeric;
procedure PEN_UP;	iniţializează starea peniţei (peniţa cu capul sus; broasca ţine capul sus);
procedure PEN_DOWN;	iniţializează starea peniţei (peniţa cu capul jos; broasca ţine capul jos);
procedure RIGHT (u: real);	schimbă direcţia broaştei spre dreapta cu u grade faţă de direcţia curentă;
procedure LEFT (u: real);	schimbă direcţia broaştei spre stînga cu u grade faţă de direcţia curentă;
procedure FOR_WARD (d :real)	broasca se deplasează înainte pe direcţia curentă pe un drum de lungime d;
procedure BACK(d);	broasca se deplasează înapoi (cu spatele) pe direcţia curentă pe un drum de lungime d;
procedure SETPOS (x,y : real) ;	poziţionează broasca în punctul de coordonate `(x,y)`, relativ faţă de poziţia curentă a broaştei;
procedure SETX (x : real) ;	poziţionează broasca în punctul de abscisă x, ordonata fiind cea curentă;
procedure SETY (y : real) ;	poziţionează broasca în punctul de ordonată y, abscisa fiind cea curentă;
procedure SET_REPER (x , y : real)	setează originea reperului în care se mişcă broasca, în punctul `(x,y)` relativ la originea iniţială (mijlocul ecranului);

6.2.1. Biblioteca TURTLE.pas

```
{---------------------------------------------------}
{          Biblioteca  TURTLE . pas                 }
{          ===========================              }
{ proceduri grafice pentru "TURTLE GRAPHICS"        }
{    autori : Roxana VLADA , Marin VLADA            }
{                                                   }
{---------------------------------------------------}
var
  penstatus               : boolean;
  cost, sint              : real;
{ penstatus :starea penitei    }
{ cost, sint : parametri  directie }
{---------------------------------------------------}
{       grafica TURTLE ( tip LOGO )                 }
{---------------------------------------------------}
{  ______________________________________________
  |                                      (90,64)|
  |            spatiul ecran                    |
  |            (dim. in mm )                    |
  |                                             |
  |                 /\                          |
  |               (0, 0)                        |
  |              pentru TURTLE                  |
  |                                             |
  |                                             |
  |(-90,-35)                          (90,-35)  |
  |_____________________________________________|
  Ymax virtual = 100 mm                         }
{{      spatiul de desen : 180 x 135 mm          }

procedure TURTLE_START;
{initializeaza modul grafic si pozitia broastei }
{(mijlocul ecranului , in directia nord )       }
begin
 DAFINI ('G');
 MOVPEN ( 1, 0.0, 0.0 ) ;
 penstatus := true ;
 cost := 0.0;
```

```
    sint := 1.0
  end;
  procedure TURTLE_STOP;
  { iesire din modul grafic      }
  begin
    DAFEN ;
  end;
  procedure PEN_UP ; { penita sus }
  { broasca tine capul sus       }
  begin penstatus := false
  end;
  procedure PEN_DOWN ; { penita jos }
  { broasca tine capul jos       }
  begin penstatus := true
  end;
  procedure RIGHT ( u:real );
  { broasca isi schimba directia spre dreapta }
  var cosu, sinu  : real;
      salv        : real;
  begin
    salv := sint ;
    cosu := cos(u * 3.141592 /180.0 ) ;
    sinu := sin(u * 3.141592 /180.0 ) ;
    sint := sint * cosu - cost * sinu ;
    cost := cost * cosu + salv * sinu
  end;
  procedure LEFT ( u:real) ;
  { broasca isi schimba directia spre stinga }
  begin
    RIGHT ( -u ) ;
  end;
  procedure FOR_WARD  ( d:real ) ;
  { broasca se deplaseaza inainte pe distanta d }
  var x,y : real ;
  begin
    x := d * cost  ;
    y := d * sint  ;
    if penstatus then MOVPEN (10, x , y )
                 else MOVPEN (00, x , y )
  end;
  procedure BACK ( d:real ) ;
  { broasca se deplaseaza inapoi( cu spatele ) }
  begin
```

```
  FOR_WARD ( -d )
end;
procedure SETPOS ( x,y : real );
{ pozitioneaza broasca in punctul (x,y)
  relativ fata de pozitia curenta a broastei }
begin
  if penstatus then MOVPEN (10, x , y )
               else MOVPEN (00, x, y )
end;
procedure SETX ( x :real );
{ pozitioneaza broasca in punctul de abscisa x si cu
  ordonata cea curenta }
begin
  SETPOS ( x , 0.0 ) ;
end;
procedure SETY ( y : real ) ;
{ pozitioneaza broasca in punctul de ordonata y si cu
  abscisa cea curenta }
begin
  SETPOS ( 0.0 , Y ) ;
end;
procedure SET_REPER ( x , y : real );
{ seteaza originea reperului fata de originea
  initiala ( mijlocul ecranului ) in punctul (x,y) }
begin
  MOVPEN ( 1 , x , y )
end;
{-------------------------------------------------}
```

Vom prezenta în continuare varianta în **TURBO PASCAL** pentru microcalculatoarele compatibile IBM–PC:

```
unit TURTLE;
{-------------------------------------------------}
{        Biblioteca  TURTLE . pas                  }
{     ******************************               }
{    ( pentru TURBO PASCAL pe IBM - PC )           }
{  proceduri grafice pentru " TURTLE GRAPHICS "    }
{                                                  }
{      AUTORI : Roxana VLADA , Marin VLADA         }
{                                                  }
{-------------------------------------------------}
interface
var
    graphdriver , graphmode        : integer;
```

```
  penstatus                    : boolean;
  cost , sint                  : real;
  ch                           : char;
{ penstatus :starea penitei    }
{ cost , sint : parametri directie }
procedure TURTLE_START;
procedure TURTLE_STOP;
procedure PEN_UP;
procedure PEN_DOWN;
procedure RIGHT(u:real);
procedure LEFT (u:real);
procedure FOR_WARD(d:real);
procedure BACK(d:real);
procedure SETPOS(x, y:real);
procedure SETX(x:real);
procedure SETY(y:real);
procedure SET_REPER(x, y:real);
implementation
uses graph, crt;
procedure TURTLE_START;
{initializeaza modul grafic si pozitia broastei }
{(mijlocul ecranului , in directia nord ) }
begin
  graphdriver := Detect ;
  initgraph(graphdriver,graphmode,' '); { mod grafic }
  setViewport( 320,170,500,200,false) ; { origine }
  setbkcolor ( 1 ) ;             { fond albastru }
  setcolor ( 4 ) ;               { desen rosu    }
  moveto ( 0 , 0 ) ;  { pozitionare in origine }
  penstatus := true;
  cost := 0.0;
  sint := 1.0
end;
procedure TURTLE_STOP;
{ iesire din modul grafic  }
begin
  ch := readkey ; { asteapta apasarea unei taste }
  closegraph ;     { iesire din modul grafic }
end;
procedure PEN_UP ; { penita sus }
{ broasca tine capul sus        }
```

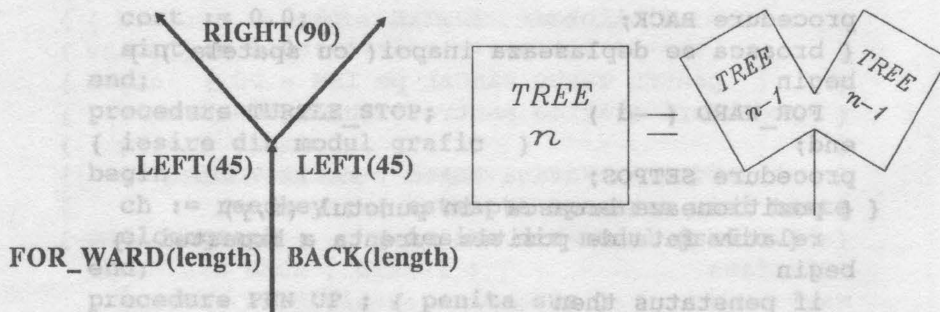
```
begin penstatus := false
end;
procedure PEN_DOWN ; { penita jos }
{ broasca tine capul jos      }
begin penstatus := true
end;
procedure RIGHT;
{ broasca isi schimba directia spre dreapta }
var cosu , sinu : real;
    salv      : real;
begin
  salv := sint ;
  cosu := cos(u * 3.141592 /180.0 ) ;
  sinu := sin(u * 3.141592 /180.0 ) ;
  sint := sint * cosu - cost * sinu ;
  cost := cost * cosu + salv * sinu
end;
procedure LEFT;
{ broasca isi schimba directia spre stinga }
begin
  RIGHT ( -u ) ;
end;
procedure FOR_WARD;
{broasca se deplaseaza inainte pe distanta d }
var x,y : real ;
begin
  x := d * cost ;
  y := d * sint ;
  if penstatus then
       lineto( getx + round(x) , gety + round(y) )
   else moveto( round(x) , round (y) )
end;
procedure BACK;
{ broasca se deplaseaza inapoi( cu spatele ) }
begin
  FOR_WARD ( -d )
end;
procedure SETPOS;
{ pozitioneaza broasca  in punctul (x,y)
  relativ fata de pozitia curenta a broastei  }
begin
  if penstatus then
        lineto( getx + round(x),gety + round(y) )
```

```
   else moverel ( round(x) ,round(y) )
end;
procedure SETX;
{ pozitioneaza broasca in punctul de abscisa x si cu
  ordonata cea curenta }
begin
  SETPOS ( x , 0.0 ) ;
end;
procedure SETY;
{ pozitioneaza broasca in punctul de ordonata y si cu
  abscisa cea curenta }
begin
  SETPOS ( 0.0 , Y ) ;
end;
procedure SET_REPER;
{ seteaza originea reperului fata de originea
  initiala ( mijlocul ecranului ) in punctul (x,y)  }
begin
  setViewport (round(x),round(y),500,200,false)
end;
{-----------------------------------------------------}
begin end.
```

6.2.2. Aplicații

Arbori binari

Ținînd seama de explicațiile date în introducerea la acest capitol, pentru desenarea unui arbore binar stilizat, traseul pe care trebuie să-l urmeze broasca prin apelarea recursivă prezentată, este următorul:



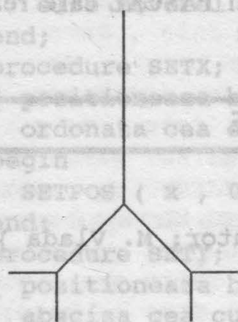
Conform acestor precizări, programul **PASCAL** care realizează desenul corespunzător are următoarea formă:

Programul TREE

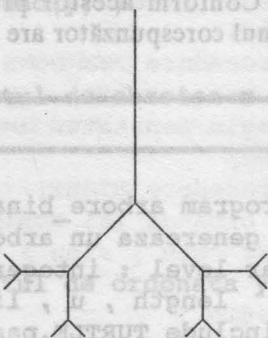
```
program arbore_binar;
{ genereaza un arbore binar -- autor: M. Vlada }
var level : integer;
    length , u , limit : real;
%include TURTLE.pas;
procedure TREE ( level : integer ; length,u : real );
begin if length < limit then  { point }
      else begin if level = 0 then { point }
                 else begin
                         FOR_WARD ( length ) ;
                         LEFT ( u );
                         TREE ( level-1, length/2);
                         RIGHT ( 2 * u ) ;
                         TREE ( level-1, length/2);
                         LEFT ( u ) ;
                         BACK ( length );
                      end
           end
end;
begin  { main }
  writeln (chr(24),' desenarea unui arbore binar');
  write ( ' originea reperului (x0,y0): ');
  read(x0,y0);
  write ( ' level : ')  ; readln ( level );
  write ( ' length : ') ; readln ( length );
  write ( ' u      : ') ; readln ( u ) ;
  write ( ' limit  : ') ; readln ( limit );
  TURTLE_START ;
  SET_REPER ( x0 , y0 ) ;
  TREE ( level , length , u ) ;
  TURTLE_STOP ;
end.
```

Varianta în **TURBO PASCAL** este următoarea:

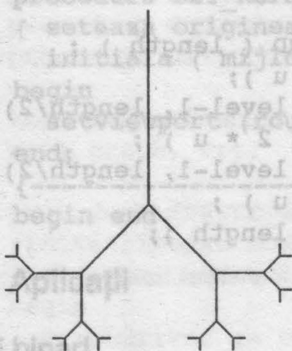
```
program arbore;
  { genereaza un arbore binar }
  { autor : ROXANA VLADA      }
  uses
```



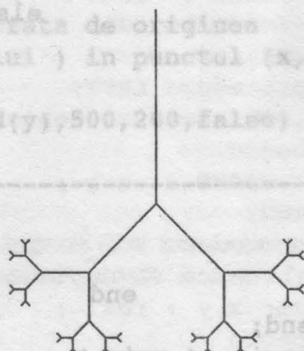
nivel=3



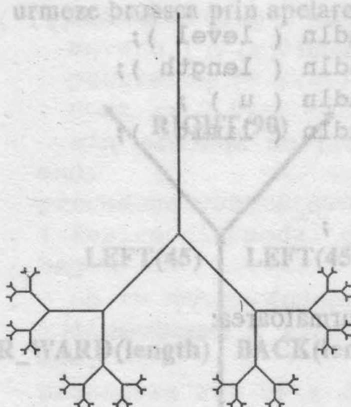
nivel=4



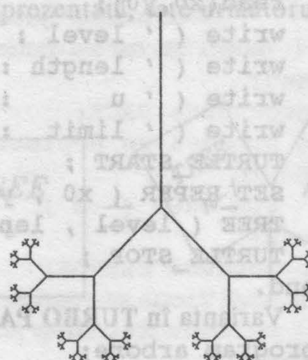
nivel=5



nivel=6



nivel=7



nivel=8

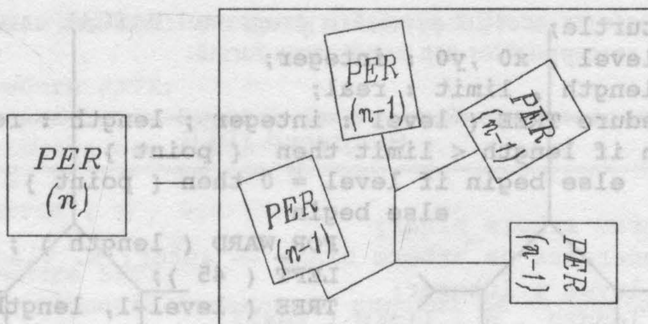
```
    turtle;
var level , x0 ,y0 : integer;
    length , limit : real;
procedure TREE ( level : integer ; length : real );
begin if length < limit then  { point }
   else begin if level = 0 then { point }
        else begin
                  FOR_WARD ( length ) ;
                  LEFT ( 45 );
                  TREE ( level-1, length/2);
                  RIGHT ( 90 ) ;
                  TREE ( level-1, length/2);
                  LEFT ( 45 ) ;
                  BACK ( length );
             end
        end
end;
begin { main }
  writeln (' desenarea unui arbore binar');
  write ( ' originea reperului (x0,y0): ');
  read(x0,y0);
  write ( ' level : '); readln ( level );
  write ( ' length : ') ; readln ( length );
  write ( ' limit  : ') ; readln ( limit );
  TURTLE_START ;
  SET_REPER ( x0 , y0 ) ;
  TREE ( level , length ) ;
  TURTLE_STOP ;
end.
```

Arborele lui PERRON

Construcția geometrică ce urmează — cunoscută sub numele de arborele lui PERRON — modelează forma unui copac. Parametrii utilizați pentru generarea figurii sînt următorii:

dim = lungimea tulpinii (prima ramură)
fdiv = factor pentru diviziunea unei ramuri
u = unghiul de ramificație dintre unele ramuri
niv = nivelul maxim de recursie

Evident, pentru generare are loc utilizarea recursivă a unei proceduri PER care trebuie să se construiască după următoarea schemă:



Programul **PASCAL** este următorul:

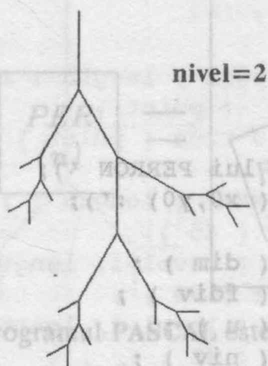
Arborele lui PERRON

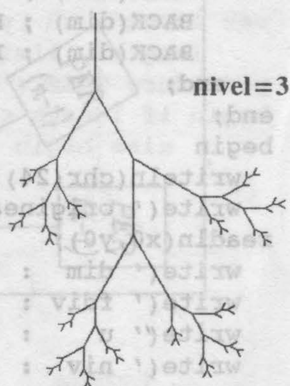
```
program arborele_PERRON;
{ genereaza arborele lui PERRON -- autor: M. Vlada }
var dim , fdiv , x0 , y0 : real ;
    niv , u              : integer ;
%include turtle.pas;
procedure PER ( dim,fdiv:real ; niv,u :integer);
{ dim  = lung. initiala ramura          }
{ fdiv = factor diviziune ptr. ramura }
{ niv  = nivelul de recursie           }
{ u    = unghi ramificatie             }
{ x0,y0 = coordonate ptr. origine      }
begin
  if niv <> 0 then
   begin
     LEFT(u) ; FOR_WARD(dim);
       PER( dim * fdiv, fdiv, niv-1, u );
     BACK(dim) ; RIGHT( 2 * u ) ;
     FOR_WARD(dim) ; RIGHT( u );
     FOR_WARD(dim) ;
        PER( dim * fdiv, fdiv, niv-1, u ) ;
     BACK(dim) ; LEFT( 2 * u ) ;
     FOR_WARD(dim) ; LEFT( u ) ;
     FOR_WARD(dim) ;
       PER( dim * fdiv, fdiv, niv-1, u ) ;
     BACK(dim) ; RIGHT( 2 * u ) ;
     FOR_WARD(dim) ;
       PER( dim * fdiv, fdiv, niv-1, u ) ;
```

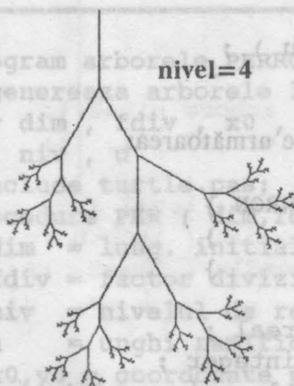
```
         BACK(dim) ; LEFT(u) ;
         BACK(dim) ; RIGHT(u) ;
         BACK(dim) ; LEFT(u)
      end;
  end;
  begin
    writeln(chr(24),' arborele lui PERRON ');
    write(' originea reperului( x0,y0) :');
  readln(x0,y0);
    write(' dim  : ') ; readln( dim ) ;
    write(' fdiv : ') ; readln( fdiv ) ;
    write(' u    : ') ; readln( u ) ;
    write(' niv  : ') ; readln( niv ) ;
    TURTLE_START ;
    SET_REPER ( x0 , y0 ) ;
    FOR_WARD(dim) ;
    PER ( dim , fdiv , niv , u ) ;
    TURTLE_STOP ;
  end.
```

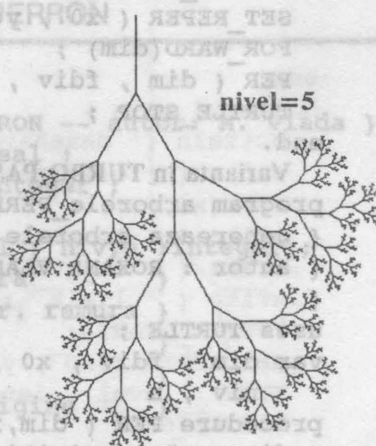
Varianta în TURBO PASCAL este următoarea:

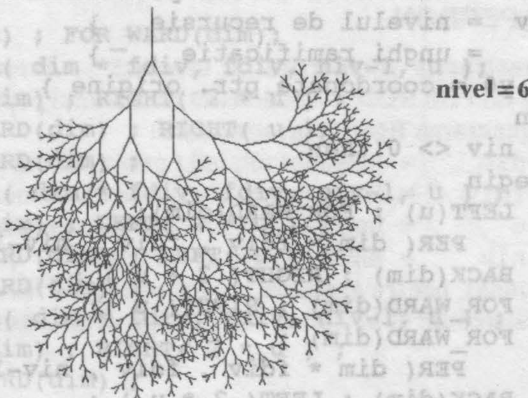
```
  program arborele_PERRON;
  { genereaza arborele lui PERRON }
  { autor : ROXANA VLADA          }
  {                               }
  uses TURTLE ;
  var dim , fdiv , x0 , y0 : real ;
      niv , u              : integer ;
  procedure PER ( dim,fdiv:real ; niv,u :integer);
  { dim  = lung. initiala ramura}
  { fdiv = factor diviziune ptr. ramura }
  { niv  = nivelul de recursie  }
  { u    = unghi ramificatie    }
  { x0,y0 = coordonate ptr. origine }
  begin
    if niv <> 0 then
      begin
         LEFT(u) ; FOR_WARD(dim);
            PER( dim * fdiv , fdiv , niv-1 , u );
         BACK(dim) ; RIGHT( 2 * u ) ;
         FOR_WARD(dim) ; RIGHT( u );
         FOR_WARD(dim) ;
            PER( dim * fdiv , fdiv , niv-1 , u ) ;
         BACK(dim) ; LEFT( 2 * u ) ;
```


nivel=2


nivel=3


nivel=4

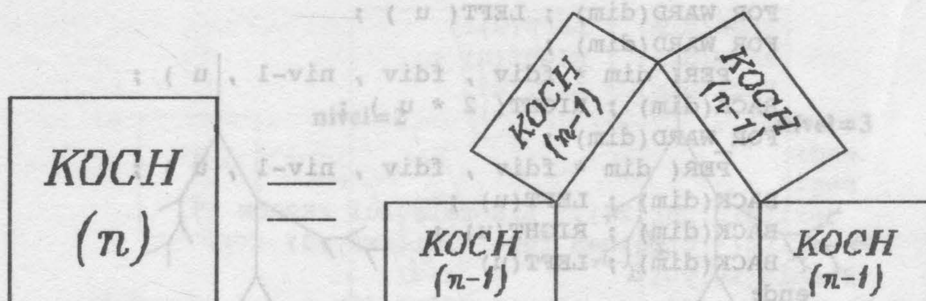

nivel=5


nivel=6

```
        FOR_WARD(dim) ; LEFT( u ) ;
        FOR_WARD(dim) ;
          PER( dim * fdiv , fdiv , niv-1 , u ) ;
        BACK(dim) ; RIGHT( 2 * u ) ;
        FOR_WARD(dim) ;
          PER( dim * fdiv , fdiv , niv-1 , u ) ;
        BACK(dim) ; LEFT(u) ;
        BACK(dim) ; RIGHT(u) ;
        BACK(dim) ; LEFT(u)
      end;
  end;
  begin
    writeln(' arborele lui PERRON ');
    write(' originea reperului( x0,y0) :');
     readln(x0,y0);
    write(' dim  : ') ; readln( dim ) ;
    write(' fdiv : ') ; readln( fdiv ) ;
    write(' u    : ') ; readln( u ) ;
    write(' niv  : ') ; readln( niv ) ;
    TURTLE_START ;
    SET_REPER ( x0 , y0 ) ;
    FOR_WARD(dim) ;
    PER ( dim , fdiv , niv , u ) ;
    TURTLE_STOP ;
  end.
```

Curba lui KOCH

Construcția geometrică ce urmează este cunoscută sub numele de curba lui KOCH și are proprietatea că este o curba infinită care delimitează o suprafață de arie finită. De asemenea, graficul curbei este un exemplu prin care se poate defini o funcție continuă dar nicăieri derivabilă; această curbă modelează un fulg de nea. Deoarece, detalii privind construcția acestei curbe fractale au fost date atît în capitolul 5, cît și în introducerea de la acest capitol, alte precizări nu sînt necesare, de aceea vom prezenta doar schema de principiu privind modul de elaborare a procedurii recursive KOCH:

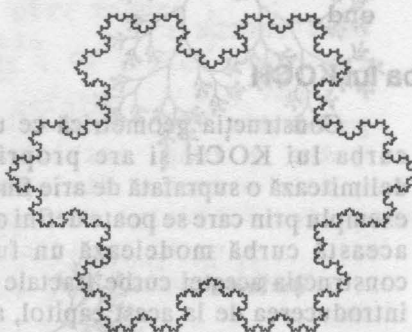


Programul care realizează construcţia curbei este:

Curba lui KOCH

```
program fulg_de_nea;
{ genereaza curba lui KOCH; autor: Rodica Niculescu }
var nivel , i      : integer;
    x0 , y0 , lat : real;
%include TURTLE.pas;
procedure KOCH ( lat, nivel : real );
{desenarea recursiva a unei laturi }
begin
```



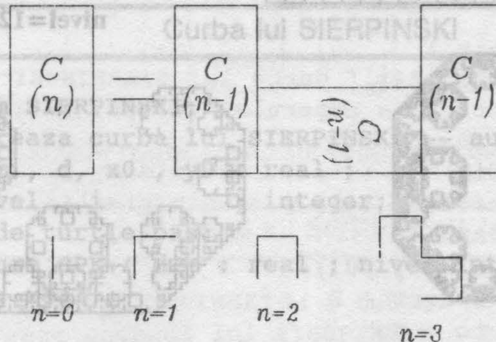


```
    if nivel = 0 then FOR_WARD ( lat )
                 else begin KOCH (lat/3, nivel-1);
                            LEFT ( 60 );
                            KOCH (lat/3, nivel-1);
                            RIGHT (120 );
                            KOCH (lat/3, nivel-1);
                            LEFT ( 60 );
```

```
                                KOCH (lat/3, nivel-1);
                           end
end;
begin { main }
   writeln(chr(24),' curba lui KOCH');
   write (' originea reperului :(x0,y0) :');
   readln(x0,y0);
   write (' lat = '); readln(lat);
   write (' nivel = '); readln(nivel);
   TURTLE_START ;
   SET_REPER ( x0 , y0 ) ;
   for i:=1 to 3 do
   begin
     KOCH ( lat, nivel );
     RIGHT ( 120 )
   end;
   TURTLE_STOP;
end.
```

Curba C

Curba C este o curbă fractală care se generează astfel: pentru nivelul n, se apelează nivelul (n-1), se schimbă direcția spre dreapta cu 90°, se apelează nivelul (n-1) și se schimbă direcția spre stînga cu 90°, adică:

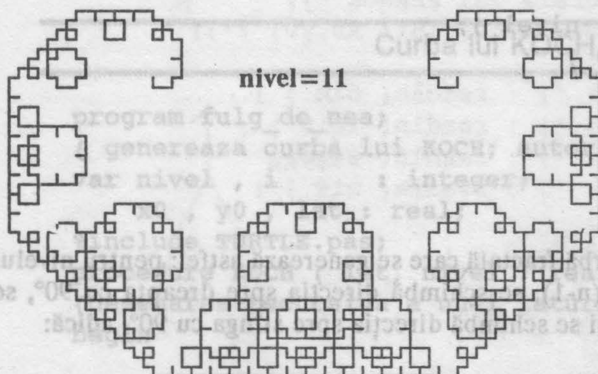


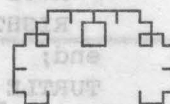
Programul care realizează costrucția curbei este:

Curba C

```
Program curbaC;
{ genereaza curba C -- autor: Florentina Hristea }
var dim , x0 , y0 : real ;
```

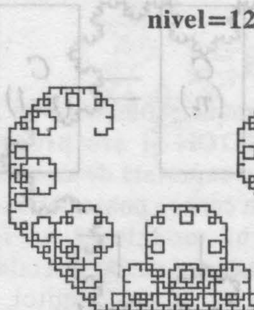
```
    nivel    : integer ;
%include turtle.pas;
procedure C ( dim, nivel : real ) ;
begin
  if nivel = 0 then FOR_WARD ( dim )
           else begin
                  C ( dim, nivel-1 ) ;
                  RIGHT ( 90 ) ;
                  C ( dim, nivel-1 ) ;
                  LEFT ( 90 ) ;
                end
end;
```









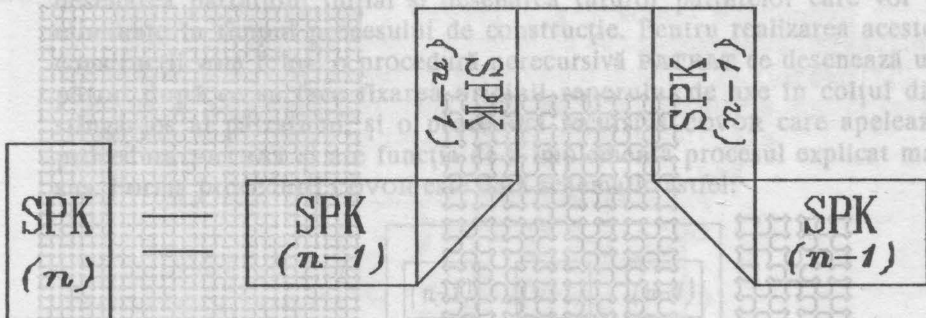



```
begin  { main }
  writeln ( chr(24),' curba C ' ) ;
  write(' originea reperului  (x0,y0) : ');
   readln(x0,y0);
  write(' dim    : ') ; readln(dim);
  write(' nivel : ') ; readln(nivel) ;
  TURTLE_START;
  SET_REPER ( x0, y0 ) ;
```

```
    C ( dim, nivel ) ;
    TURTLE_STOP
end.
```

Curba lui SIERPINSKI

Curba lui SIERPINSKI are proprietatea că — la infinit — este o curbă de lungime infinită delimitată de o suprafață de arie finită, și anume de un pătrat. Procedura `SPK` ce realizează construcția acestei curbe se definește astfel:

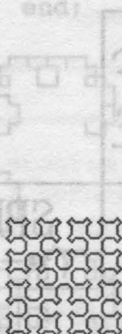


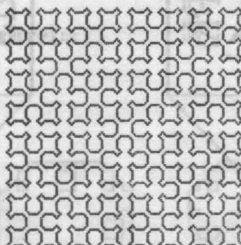
Programul care realizează construcția curbei este:

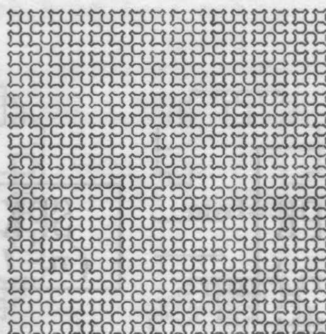
Curba lui SIERPINSKI

```
program SIERPINSKI;
{ genereaza curba lui SIERPINSKI -- autor: M. Vlada }
var dim , d, x0 , y0 : real ;
    nivel , i        : integer;
%include turtle.pas;
procedure SPK ( dim : real ; nivel:integer );
var d : real ;
begin
  if nivel <> 0 then
                begin
                  d := dim / sqrt(2);
                     SPK( dim, nivel-1 );
                  RIGHT (45);
                  FOR_WARD ( d );
                  RIGHT (45);
                        SPK ( dim, nivel-1 );
```

```
                    LEFT ( 90 );
                    FOR_WARD ( dim );
                    LEFT ( 90 );
                        SPK ( dim, nivel-1 );
                    RIGHT ( 45);
                    FOR_WARD ( d );
                    RIGHT ( 45 );
                        SPK ( dim, nivel-1);
                      end
   end;
```


nivel=3

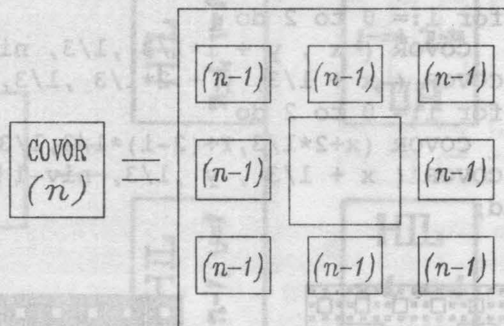

nivel=4


nivel=5

```
   begin  { main }
     writeln (chr(24),' curba lui SIERPINSKI');
     write(' originea reperului (x0,y0) : ');
      readln (x0,y0);
     write(' dim       : ') ; readln(dim) ;
     write(' nivel     : ') ; readln(nivel);
     TURTLE_START;
     SET_REPER ( x0, y0 );
     d := dim / sqrt ( 2 );
     for i:=1 to 4 do
       begin
         SPK ( dim, nivel ) ;
         RIGHT ( 45 ) ;
         FOR_WARD ( d );
         RIGHT ( 45 )
       end;
     TURTLE_STOP
   end.
```

Covorul lui SIERPINSKI

Covorul lui SIERPINSKI reprezintă un exemplu de obiect geometric despre care nu s-a putut preciza dacă este o curbă ori o suprafață. Costrucția este simplă, și anume, se pleacă de la un pătrat a cărui suprafață se împarte în 9 părți egale prin împărțirea fiecărei laturi în cîte 3 părți egale. Partea din mijlocul pătratului inițial se elimină, iar cele 8 pătrate rămase sînt supuse aceluiași procedeu. Prin continuarea acestui proces, care se aplică pătratelor care nu se elimină, la infinit, se obține o construcție geometrică despre care nu se poate spune că reprezintă o curbă ori o suprafață. Reprezentarea grafică a acestui proces o vom face prin desenarea pătratului inițial și desenarea tuturor pătratelor care vor fi eliminate în timpul procesului de construcție. Pentru realizarea acestei construcții vom folosi o procedură nerecursivă PATRAT ce desenează un pătrat după ce se face fixarea originii reperului de axe în colțul din stînga-jos al pătratului, și o procedură recursivă COVOR care apelează procedura PATRAT și are funcția de a implementa procesul explicat mai sus. Forma procedurii COVOR este dată schematic astfel:

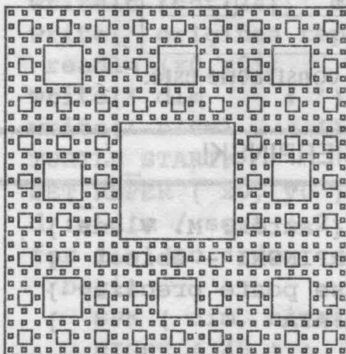


Programul care realizează acestă construcție este:

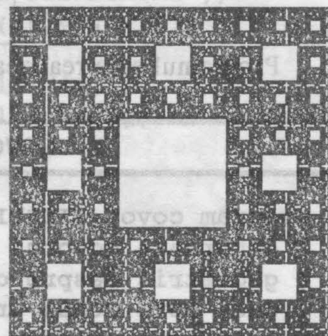
Covorul lui SIERPINSKI

```
program covor_SIERPINSKI;  { autor: M. Vlada }
{ genereaza covorul lui SIERPINSKI - obiect  }
{ geometric despre care nu se poate preciza  }
{ daca este CURBA ori SUPRAFATA              }
var x0,y0,l : real;
    niv     : integer ;
%include turtle.pas;
procedure PATRAT ( x,y,l : real );
begin
  PEN_UP ;
```

```
    SET_REPER ( x,y ) ;
    PEN_DOWN ;
    FOR_WARD (l) ;
    RIGHT (90);
    FOR_WARD (l) ;
    RIGHT (90) ;
    FOR_WARD (l) ;
    RIGHT (90) ;
    FOR_WARD (l) ;
    RIGHT (90)
  end;
  procedure COVOR ( x,y,l:real ; niv :integer );
  var i :integer ;
  begin
    if niv = 0 then PATRAT ( x + l/3,y+ l/3 , l/3 )
       else
         begin
           PATRAT( x + l/3 , y + l/3 , l/3 );
           for i:= 0 to 2 do
             COVOR ( x , y + i*l/3 ,l/3, niv-1 ) ;
           COVOR ( x + l/3, y + 2*l/3 ,l/3, niv-1);
           for i:= 0 to 2 do
             COVOR (x+2*l/3,Y+(2-i)*l/3,l/3,niv-1);
           COVOR ( x + l/3 , y ,l/3, niv-1 );
         end;
  end;
```



nivel=3



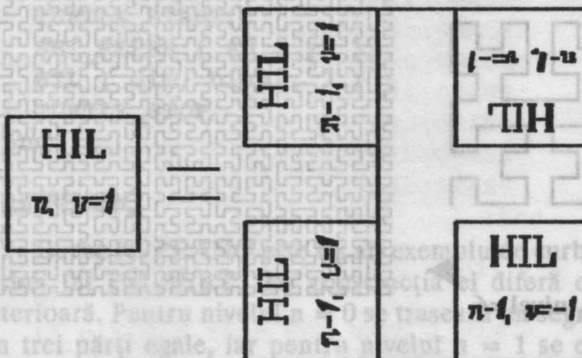
nivel=4

```
  begin
    write(chr(24),' covorul lui SIERPINSKI ');
    write(' originea reperului (x0,y0) : ');
```

```
  readln(x0,y0);
  write(' lungime latura   : ');readln(l);
  write(' nivel : '); readln(niv) ;
  TURTLE_START ;
  SET_REPER ( x0,y0) ;
  PATRAT ( x0,y0,l );
  COVOR ( x0,y0,l,niv) ;
  TURTLE_STOP ;
end.
```

Curba lui HILBERT

Curba lui HILBERT este un exemplu de curbă continuă de lungime infinită care „umple" un pătrat, adică este o funcție $f:[0,1] \to [0,1]\times[0,1]$ continuă și surjectivă, dar neinjectivă. Pentru construcția acestei curbe vom defini procedura HIL care se proiectează astfel:

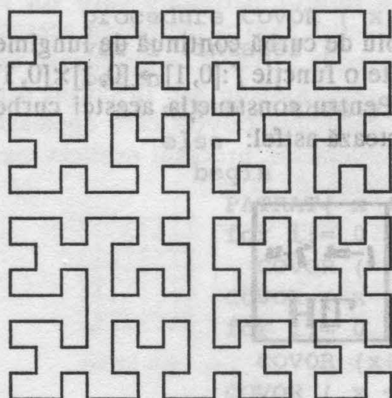


Programul care realizează această construcție este:

Curba lui HILBERT

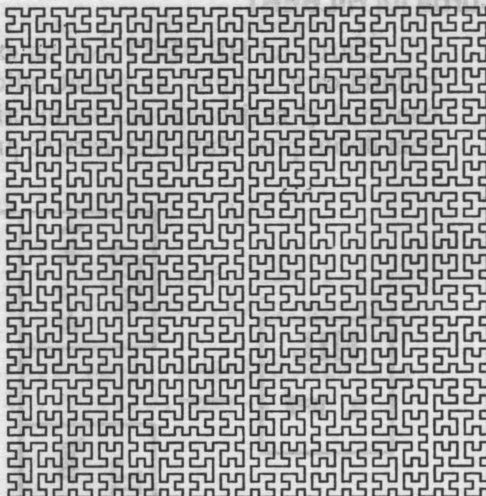
```
program HILBERT;
{ genereaza curba lui HILBERT -- autor: M. Vlada }
var dim , x0 , y0  : real ;
    v   , nivel    : integer ;
%include turtle.pas;
procedure HIL( dim  :real ; nivel , v : integer );
begin
  if nivel <> 0 then
    begin
      LEFT ( v * 90 );
```

nivel=1 **nivel=2** **nivel=3**



nivel=4

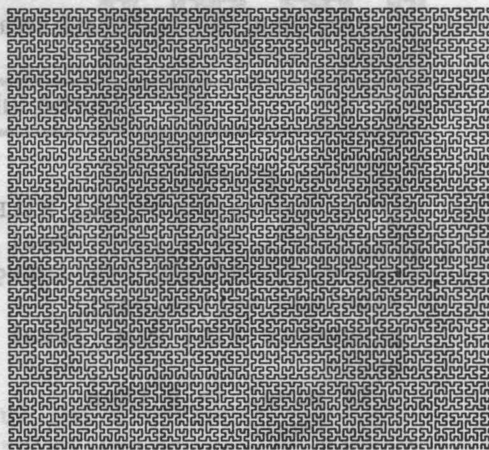




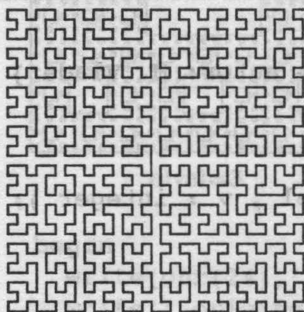
nivel=6

nivel=7





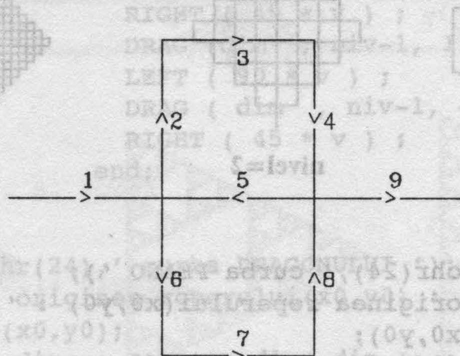
nivel=5



```
        HIL ( dim, nivel-1, - v );
        FOR_WARD ( dim );
        RIGHT ( v * 90 );
        HIL ( dim, nivel-1, v );
        FOR_WARD ( dim );
        HIL ( dim, nivel-1, v );
        RIGHT ( v * 90 );
        FOR_WARD ( dim );
        HIL ( dim, nivel-1, - v );
        LEFT ( v * 90 );
      end
end;
begin { main }
  writeln ( chr(24),' curba lui HILBERT ');
  write (' originea reperului (x0,y0) :');read(x0,y0);
  write ( ' dim   : '); readln ( dim );
  write ( ' nivel : '); readln ( nivel );
  TURTLE_START;
  SET_REPER ( x0 , y0 );
  HIL ( dim, nivel, 1 ) ;
  TURTLE_STOP
end.
```

Curba lui PEANO

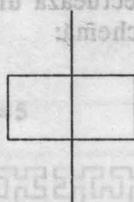
Curba lui PEANO este un alt exemplu de curbă care are proprietatea curbei lui HILBERT, dar construcţia ei diferă esenţial de construcţia anterioară. Pentru nivelul n = 0 se trasează un segment de dreaptă format din trei părţi egale, iar pentru nivelul n = 1 se efectuează următoarele trasări în ordinea indicată de numerele ce apar pe schemă:



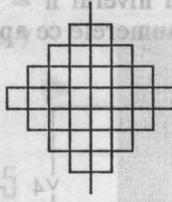
Programul care realizează această construcție este:

Curba lui PEANO

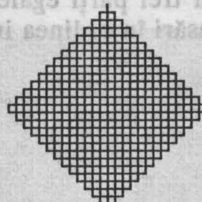
```
program PEANO;
{ genereaza curba lui PEANO -- autor: M. Vlada }
var dim , x0 ,y0 : real ;
    niv : integer ;
%include turtle.pas;
procedure PEANO ( dim : real ; niv : integer ) ;
begin
  if niv = 0 then FOR_WARD ( dim )
           else
            begin
              PEANO(dim/3,niv-1) ; LEFT(90) ;
              PEANO(dim/3,niv-1) ; RIGHT(90);
              PEANO(dim/3,niv-1) ; RIGHT(90);
              PEANO(dim/3,niv-1) ; RIGHT(90);
              PEANO(dim/3,niv-1) ; LEFT(90) ;
              PEANO(dim/3,niv-1) ; LEFT(90) ;
              PEANO(dim/3,niv-1) ; LEFT(90) ;
              PEANO(dim/3,niv-1) ; RIGHT(90) ;
              PEANO(dim/3,niv-1)
            end;
end;
```



nivel=1



nivel=2



nivel=3

```
begin
  writeln(chr(24),' curba PEANO ');
  write(' originea reperului(x0,y0) : ');
   readln(x0,y0);
  write(' dim : ') ; readln( dim ) ;
  write(' niv : ') ; readln( niv ) ;
```

```
    TURTLE_START ;
    SET_REPER ( x0,y0 ) ;
    PEANO ( dim , niv ) ;
    TURTLE_STOP ;
  end.
```

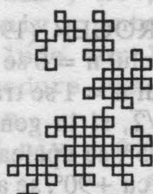
Curba Dragonului

Curba Dragonului (GARDNER (1980), KROGER (1984)) este o curbă care se generează recursiv astfel: pentru nivelul n = 0 se trasează un segment de dreaptă de lungime dim, pentru nivelul n = 1 se trasează două laturi alăturate ale unui pătrat de latură dim/$\sqrt{2}$, și în general pentru nivelul n, se schimbă direcția spre dreapta cu +45°, se apelează procesul pentru nivelul n-1, se schimbă direcția spre stînga cu +90°, se apelează din nou procesul pentru nivelul n-1 și în final, se schimbă direcția spre dreapta cu +45°. Programul care realizează această construcție are următoarea formă:

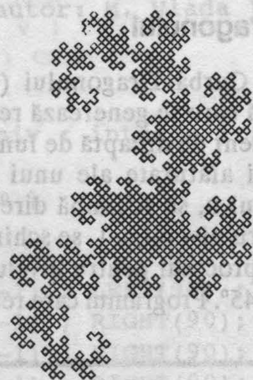
Curba Dragonului

```
program DRAGON;
{ genereaza curba dragonului -- autor: M. Vlada }
var dim , x0,y0 : real ;
    niv , v     : integer;
%include turtle.pas;
procedure DRAG ( dim :real ; niv , v : integer );
begin
  if niv = 0 then FOR_WARD ( dim )
             else
               begin
                 RIGHT ( 45 * v ) ;
                 DRAG (dim   , niv-1, 1 );
                 LEFT ( 90 * v ) ;
                 DRAG ( dim   , niv-1, -1);
                 RIGHT ( 45 * v ) ;
               end;
end;
begin
  write(chr(24),' curba DRAGONULUI ');
  write(' originea reperului(x0,y0) : ');
   readln(x0,y0);
  write(' dim  : ') ; readln( dim );
  write(' niv  : ') ; readln( niv ) ;
```

```
    TURTLE_START ;
    SET_REPER (x0,y0);
    DRAG ( dim /sqrt(2) , niv , 1 );
    TURTLE_STOP ;
end.
```



nivel=8

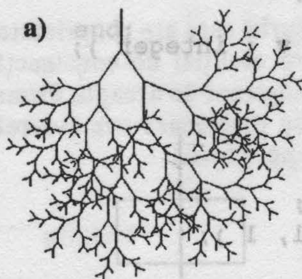


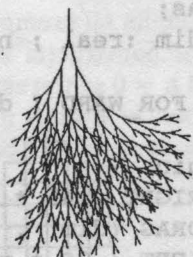
nivel=11

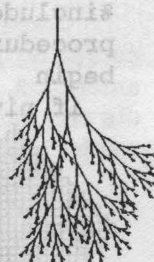
6.2.3. Probleme propuse

Să se elaboreze algoritmii, procedurile şi programele corespunzătoare pentru realizarea următoarelor construcţii geometrice:

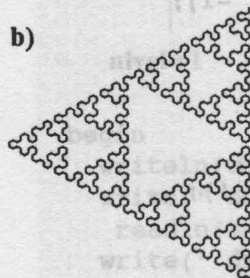
a)

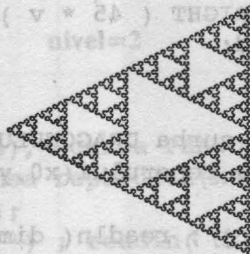


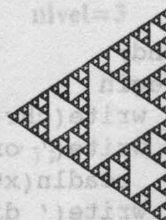




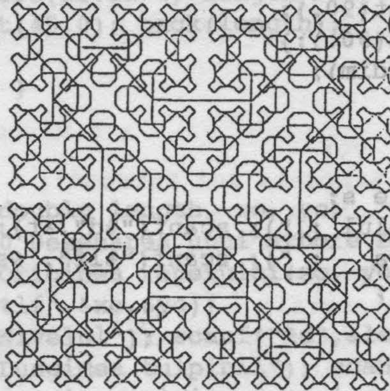
b)







c)



6.3. Grafica TURTLE în limbajul C

Această secțiune există numai pentru simetria capitolului 6. Ea este doar o adaptare (utilă măcar pentru comparații) a secțiunii 6.2.

6.3.1. Biblioteca TURTLE.c

Sursele funcțiilor date (chiar în exces) la programul SIERP (din capitolul 1) constituie baza de plecare pentru realizarea de T–grafică în C. Ele se pot folosi fie pentru incluziune lexicală, fie pentru obținerea unei biblioteci de module obiect (în accepțiunea RSX–11M). Vom observa doar că desenele au fost executate pe masa de desen DGF–1712.

6.3.2. Aplicații

Curba C

```
/*
    Curba C

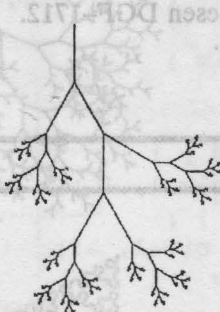
    autor: A. Posea
*/
$$narg=1;
static double dim=5.;
C(t)
int t;
{
if (t) {
```

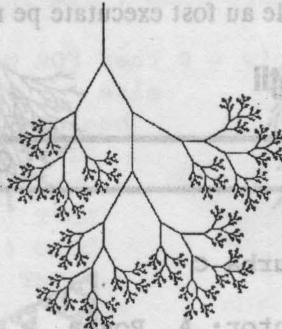
```
C(t-1); right(90.);
C(t-1); left(90.);}
else forwrd(dim);
}
main()
{
int t; double s;
printf("Nivelul : "); scanf("%d",&t);
printf("Scara   : "); scanf("%lf", &s);
trtl_start();
scale(s, s);
C(t);
trtl_stop();
}
#include "turtle.c";
```

Arbore binar

```
/*
   Arbore binar

   autor: A. Posea
*/
static double limit;
```





```
arbore(level, length)
int level; double length;
{ if (length > limit) {
    if (level) {
        forwrd(length); left(45.0);
        arbore(level-1, length/2.0);
        right(90.0);
```

```
        arbore(level-1, length/2.0);
        left(45.0); back(length);
        }
      }
}
main()
{
int level; double length, xo, yo;
printf("\030 Desenarea unui arbore binar");
printf("\n Originea reperului(xo, yo)";
scanf("%lf,%lf",&xo,&yo);
printf("\n Nivelul"); scanf("%d",&level);
printf("\n Lungimea tulpinii"); scanf("%lf",&length);
printf("\n Lungimea minima a unei ramuri");
scanf("%ld",&limit);
trtl_start(); set_reper(xo,yo);
    arbore(level, length);
trtl_stop();
}
```

Arborele lui PERRON

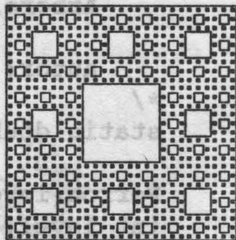
```
/*
    Arborele lui Perron

    autor: A. Posea
*/
static double mugure, unghi;

Perron(level, ramura)
int level; double ramura, unghi;
{ if (level) {
    left(unghi); forwrd(ramura);
    Perron(level-1, ramura*mugure);
    back(ramura); right(unghi*2.0); forwrd(ramura);
    Perron(level-1, ramura*mugure);
    back(ramura); left(unghi*2.); forwrd(ramura);
    left(unghi); forwrd(ramura);
    Perron(level-1, ramura*mugure);
    back(ramura); right(unghi*2.0); forwrd(ramura);
    Perron(level-1, ramura*mugure);
    back(ramura); left(unghi);
```

```
        back(ramura); right(unghi);
        back(ramura); left(unghi);
        }
}
main()
{ double xo,  yo;
printf("\030 Arborele lui Perron");
printf("\n Originea reperului: ");
scanf("%lf,%lf",&xo,&yo);
printf("\n Lungimea tulpinii: ");
scanf("%lf",&ramura);
printf("\n Nivelul: "); scanf("%d",&level);
printf("\n Pozitia mugurelui (0<p<1, u): ");
scanf("%lf,%lf",&mugure,&unghi);
trtl_start(); set_reper(xo,yo);
    forwrd(ramura); Perron(level, ramura);
trtl_stop()
}
```

Covorul lui SIERPINSKI



```
/*
    Covorul lui Sierpinski

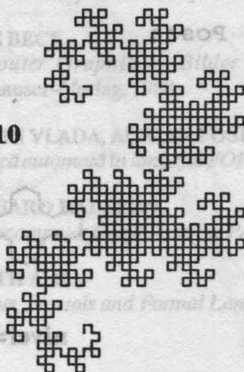
    autor: A. Posea
*/
patrat(xo,yo,latura)
double xo,yo,latura;
{ register int i;
setrep(xo, yo);
 for(i=4; i; i--){
 forwrd(latura); right(90.0);}

}
covor(level, xo, yo, latura)
int level; double xo, yo, latura;
{ double lat; register int i; lat = latura/3.;
patrat(xo+lat, yo+lat, lat);
if (level) {
    for (i=0; i<3; i++)
        covor(level-1, xo, yo+i*lat, lat);
    covor(level-1, xo+lat, yo+2.*lat, lat);
    for (i=3; i>0; i--)
```

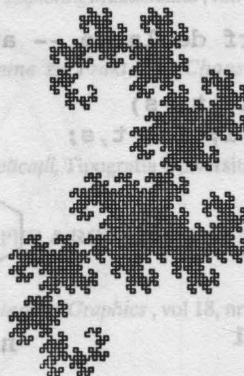
```
        covor(level-1, xo+2.*lat, yo+(i-1)*lat, lat);
    covor(level-1, xo+lat, yo, lat);
     }
}
#include "turtle.c";
$$narg=1;
main()
{
double latura; int level;
printf("\030 Lungime latura : ");
scanf("%lf",&latura);
printf("\n Nivelul : "); scanf("%d",&level);
start();
patrat (0.,0.,latura);
covor(level,0.,0.,latura);
stop();
}
```

Curba Dragonului

```
/*
    Curba Dragonului -- autor: A. Posea
*/
static double length=10.0;
dragon(virsta, sens)
```



nivel=10



nivel=12

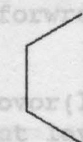
```
int virsta, sens;
{
if (virsta)
    { right(45.*sens);
```

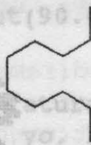
```
        dragon(virsta-1, sens);
        left(90.*sens);
        dragon(virsta-1, -sens);
        right(45.*sens);
        } else forwrd(length);
}
main()
{
int virsta;
double sc;
printf("Virsta Dragonului este "); scanf("%d",
&virsta);
printf("Factorul de scara este "); scanf("%lf", &sc);
start();
scale(sc,sc);
dragon(virsta, 1);
stop();
}
#include "turtle.c";
```

Vom prezenta un program care nu este translatare în C a unuia din 6.2.2 cu recomandarea de a descoperi singuri iniţiatorul şi generatorul figurii rezultate, încercînd eventual unele modificări:

Programul Vîrf de Lance

```
/*
        Virf de lance -- autor: A. Posea
*/
LANCE(l, t, s)
double l; int t,s;
```


nivel=1


nivel=2


nivel=3

```
{
if (t) {
        left(60.*s);
        LANCE(l*0.5, t-1, -s); right(60.*s);
        LANCE(l*0.5, t-1, s); right(60.*s);
```

```
        LANCE(l*0.5, t-1, -s); left(60.*s);}
    else
        forwrd(l);
    }
    $$narg=1;
    main()
    {
    double l; int t;
    printf("Virsta : "); scanf("%d", &t);
    printf("Dimens : "); scanf("%lf", &l);
    trtl_start();
    LANCE(l, t, 1);
    trtl_stop();
    }
    #include "turtle.c";
```

6.3.3. Probleme propuse

1) Să se realizeze unele variaţiuni pe temele fulg-de-nea, curba lui Hilbert, curba lui Sierpinski.

2) Să se calculeze (în funcţie de t), orientarea iniţială a broaştei, astfel ca figura realizată de procedura C să aibă orientarea literei „C".

3) Să se calculeze (în funcţie de t) lungimea curbei desenate de procedura LANCE.

6.4. Bibliografie

[1] HAROLD ABELSON, ANDREA A. DISESSA,
Turtle Geometry — The Computer as a Medium for Exploring Mathematics, vol. I–II, The MIT Press, 1981.

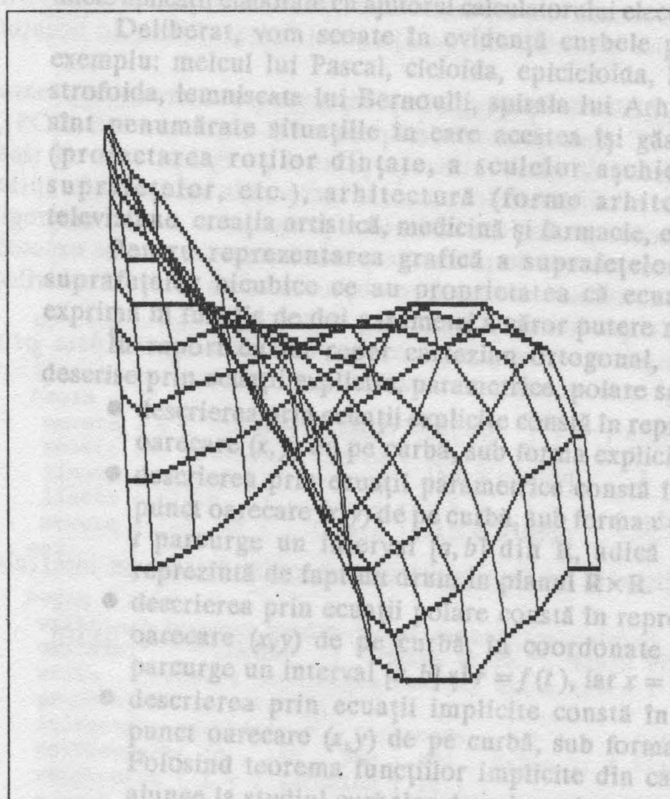
[2] UWE BECK,
Computer Graphik — Bilder und Programme zu Fraktalen, Chaos und Selbstähnlichkeit, Birkhauser-Verlag, 1988.

[3] MARIN VLADA, ADRIAN POSEA,
Grafică automată în limbajul FORTRAN 77 şi aplicaţii, Tipografia Universităţii Bucureşti, ed. II, 1990.

[4] RICHARD HALPERN,
Microcomputer Graphics using PASCAL, HARPER & ROW, Publishers, New York, 1985.

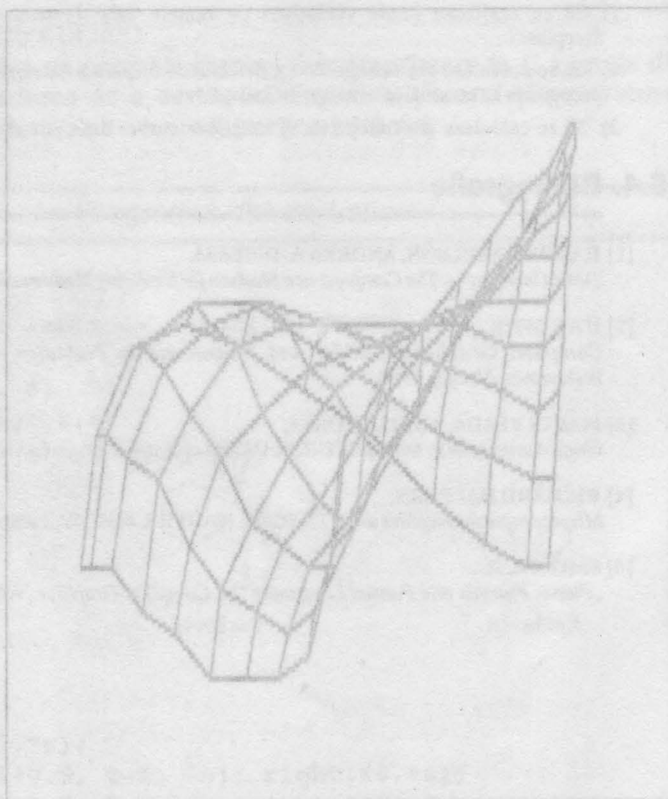
[5] SMITH A. R.,
„Plants, Fractals and Formal Languages" în: *Computer Graphics*, vol 18, nr. 3, 1984.

APLICAȚII ÎN TEORIA CURBELOR ȘI SUPRAFEȚELOR



7

7. APLICAŢII ÎN TEORIA CURBELOR ŞI SUPRAFEŢELOR . 135



7

7

Reprezentarea grafică a curbelor şi suprafeţelor reclamă cîteva probleme a căror rezolvare depinde de stăpînirea unor noţiuni atît din domeniul matematicii cît şi din domeniul informaticii. De aceea, în acest capitol vom trata cîteva exemple semnificative din domeniul curbelor şi suprafeţelor, dar şi pentru a familiariza pe cititor cu complexitatea acestui domeniu de mare interes atît pentru matematicieni cît şi pentru ingineri sau informaticieni. Formele unor curbe şi suprafeţe sînt dificil de reprezentat, dar acestea sînt deosebit de utile în unele aplicaţii elaborate cu ajutorul calculatorului electronic.

Deliberat, vom scoate în evidenţă curbele plane remarcabile (de exemplu: melcul lui Pascal, cicloida, epicicloida, hipocicloida, astroida, strofoida, lemniscata lui Bernoulli, spirala lui Arhimede, etc.), deoarece sînt nenumărate situaţiile în care acestea îşi găsesc aplicaţii: tehnică (proiectarea roţilor dinţate, a sculelor aşchietoare, desfăşurarea suprafeţelor, etc.), arhitectură (forme arhitecturale deosebite), televiziune, creaţia artistică, medicină şi farmacie, etc.

Pentru reprezentarea grafică a suprafeţelor vom folosi metoda suprafeţelor bicubice ce au proprietatea că ecuaţiile parametrice se exprimă în funcţie de doi parametri a căror putere nu depăşeşte pe 3.

În raport cu un reper cartezian ortogonal, curbele plane pot fi descrise prin ecuaţii explicite, parametrice, polare sau implicite:

- descrierea prin ecuaţii explicite constă în reprezentarea unui punct oarecare (x,y) de pe curbă, sub forma explicită $y = f(x)$.
- descrierea prin ecuaţii parametrice constă în reprezentarea unui punct oarecare (x,y) de pe curbă, sub forma $x = f(t), y = g(t)$, unde t parcurge un interval $[a,b]$ din R, adică $F(t) = (f(t), g(t))$ ce reprezintă de fapt un drum în planul R×R.
- descrierea prin ecuaţii polare constă în reprezentarea unui punct oarecare (x,y) de pe curbă, în coordonate polare (r,t), unde t parcurge un interval $[a,b]$ şi $r = f(t)$, iar $x = r\cos(t), y = r\sin(t)$.
- descrierea prin ecuaţii implicite constă în reprezentarea unui punct oarecare (x,y) de pe curbă, sub forma ecuaţiei $F(x,y) = 0$. Folosind teorema funcţiilor implicite din calculul diferenţial, se ajunge la studiul curbelor descrise prin ecuaţii explicite.

În toate cazurile, dacă sînt îndeplinite condiţii corespunzătoare din calculul diferenţial, se ajunge la studiul curbelor descrise prin ecuaţii explicite. După cum vom vedea, această trecere trebuie uneori evitată.

7.1. Curbe date de ecuaţii explicite

În acest caz, reprezentarea curbelor plane este dată de expresia **explicită** a coordonatei y funcţie de coordonata x, adică $y=f(x)$. Reprezentarea grafică cu ajutorul calculatorului a acestor curbe se realizează prin parcurgerea domeniului $[a,b]$ al abscisei x şi determinarea valorilor $f(x)$, $x \in [a,b]$ ţinînd seama de unele scalări atît pe Ox cît şi pe Oy în funcţie de particularităţile funcţiei f. Pentru implementarea pe calculator a acestei reprezentări trebuie să se ţină seama de dispozitivul periferic de afişare grafică pe care utilizatorul doreşte să execute imaginea. Pentru minicalculatoarele compatibile PDP (sub sistemul de operare RSX-11M), în capitolele 1 şi 2 din volumul I s-au prezentat două biblioteci de proceduri grafice utilizabile din limbajele **PASCAL** şi **C** care conţin fiecare procedura corespunzătoare (procedura GRAPH) pentru reprezentarea grafică pe intervalul $[a,b]$ a unei funcţii f date de ecuaţia explicită $y=f(x)$.

În limbajul **TURBO PASCAL** implementat pe microcalculatoare compatibile IBM-PC (sub sistemul de operare PC-DOS sau MS-DOS), este necesară scrierea de către utilizator a unei proceduri care să ţină seama că în acest caz coordonatele cu care se lucrează în modul grafic (prin utilizarea procedurilor grafice din biblioteca `graph`) sînt numere întregi. Din acest motiv este nevoie de precauţie privind scara la care se execută desenul. Scalarea ce trebuie realizată se obţine în urma analizei valorilor extreme ale funcţiei pe domeniul pe care se reprezintă.

Pentru exemplificare, vom reprezenta grafic trei funcţii date prin ecuaţii explicite şi care au anumite caracteristici:

- $f(x) = |\sin x| \cdot e^{-\sin x}$ pe intervalul $[0, 8\pi]$;
- $f(x) = \begin{cases} 1, & \text{pentru } x = 0 \\ \dfrac{d(x)}{x}, & \text{pentru } x \geq 0, \end{cases}$

unde $d(x)$ este distanţa de la x la cel mai apropiat întreg, pe intervalul [0,100];

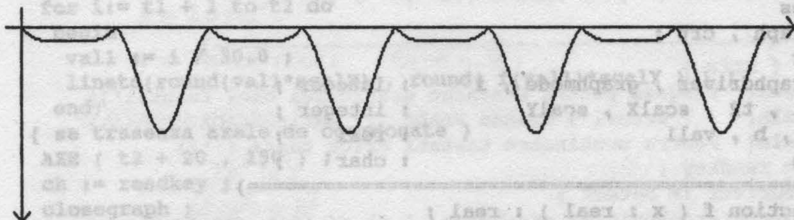
- $f(x) = 5 \cdot e^{-0,3x} + 15 \cdot e^{-0,1x} - 20 \cdot e^{-0,2x}$, pe intervalul [0,10].

EXEMPLUL 1:

Programul GRAFIC1

```
{=======================================================}
program GRAFIC1 ;
{   autori  : Roxana VLADA , Marin VLADA           }
uses
 graph , crt ;
var
 graphdriver , graphmode , i   : integer ;
 ch                            : char    ;
 a , b , vall                  : real    ;
 t1 ,t2 , scalX , scalY        : integer ;
{=======================================================}
function f ( x: real ) : real ;
{------------------------------}
begin
 f := abs ( sin(x) ) * exp ( - sin ( x ) ) ;

end;
{----------------------------------------}
```



```
procedure AXE ( t1 , t2 : integer ) ;
{----------------------------------}
{ traseaza axele de coordonate     }
begin
 moveto ( 0 , -10 ) ; lineto ( 0 , t2  ) ;
 moveto (-5 , t2-5) ; lineto ( 0 , t2  ) ;
 lineto (+5 , t2-5) ; moveto (-10, 0   ) ;
 lineto ( t1, 0   ) ; lineto (t1-5,5   ) ;
 moveto (t1-5, -5 ) ; lineto ( t1, 0   ) ;
end;
{----------------------------------------}
begin { main }
 writeln(' Program pentru trasarea graficului unei functii ');
 writeln('     autori   :  ROXANA VLADA  , MARIN VLADA      ');
 write  (' Pentru continuare tastati orice tasta ! '); read(ch);
 graphdriver := Detect ;
 initgraph( graphdriver , graphmode , ' ' ) ; { init mod grafic }
 setViewport( 20 , 150 , 500 , 300 , false ); { fixare origine. }
 setbkcolor ( 1 ) ; setcolor ( 14 ) ;
 a := 0.0  ; b := 8.0 * pi ;
```

```
  { se traseaza graficul  }
  scalX := 20  ;  scalY := 20 ; { scalare pe X si Y }
  t1 := round(a) * scalX ; t2 := round(b) * scalX ;
  moveto ( round(a) * scalX , round( f(a) * scalY ) ) ;
  for i:= t1 + 1  to t2 do
   begin
    val1 := i / 20.0 ;
    lineto ( round(val1 * scalX)  , round( f(val1) * scalY )  ) ;
   end;
  { se traseaza axele de coordonate }
  AXE ( t2+20 , 100 ) ;
  ch := readkey ;
  closegraph ;
  end.
```

EXEMPLUL 2:

Programul GRAFIC2

```
{=========================================}
program GRAFIC2;
{  autori : Roxana Vlada , Marin Vlada          }
uses
 graph , crt ;
var
 graphdriver , graphmode , i      : integer ;
 t1 , t2 , scalX , scalY          : integer ;
 a , b , val1                     : real     ;
 ch                               : char     ;
{=========================================}
function f ( x : real ) : real ;
{-------------------------------}
begin
 if x=0.0 then
  f:= 1.0
        else
  f := abs( round( x )-x ) / x ;
end;
{-------------------------------}
procedure AXE ( t1 , t2 : integer ) ;
```

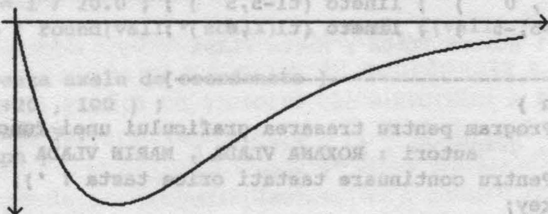
```
{---------------------------------------}
{ traseaza axele de coordonate        }
begin
 moveto ( 0 , -10 )  ; lineto ( 0 , t2 ) ;
 moveto (-5 , t2-5)  ; lineto ( 0 , t2 ) ;
 lineto (+5 , t2-5)  ; moveto (-10, 0  ) ;
 lineto (t1 , 0   )  ; lineto (t1-5,5  ) ;
 moveto (t1-5,-5  )  ; lineto (t1  ,0  ) ;
end;
{---------------------------------------}
begin { main }
 writeln(' Program pentru trasarea graficului unei functii');
 writeln('         autori : ROXANA VLADA , MARIN VLADA       ');
 write  (' Pentru continuare tastati orice tasta ! ');
 ch := readkey;
 graphdriver := Detect ;
 initgraph ( graphdriver , graphmode ,' ') ;{init grafic mod}
 setVieWport( 20 , 150 , 500 , 300 ,false) ;{fixare origine }
 setbkcolor ( 1 ) ; setcolor ( 14 ) ;
 a := 0.0  ; b := 15.0 ;
{ se traseaza graficul }
 scalX := 30 ; scalY := 140 ; { scalare pe X si Y }
 t1 := round (a) * scalX ; t2 := round (b) * scalX ;
 moveto( t1 , round( f(a) * scalY ) ) ;
 for i:= t1 + 1 to t2 do
  begin
   val1 := i / 30.0 ;
   lineto(round(val1*scalX) , round( f(val1)*scalY ) ) ;
  end;
{ se traseaza axele de coordonate }
 AXE ( t2 + 20 , 150 ) ;
 ch := readkey ;
 closegraph ;
end.
```

EXEMPLUL 3:

Programul GRAFIC3

```
{==============================================}
program GRAFIC3;
{   autori  :  ROXANA VLADA , MARIN VLADA       }
uses
 graph , crt ;
var
 graphdriver , graphmode , i     : integer ;
 t1 , t2 , scalX , scalY         : integer ;
 a , b , val1                    : real    ;
 ch                              : char    ;
{==============================================}
function f ( x : real ) : real ;
```

```
{--------------------------}
{ functia "absortie-eliminare" }
begin
 f := 5.0 * exp ( -0.3*x ) + 15.0 * exp( -0.1*x ) -
     20.0 * exp ( -0.2 * x ) ;
end;
```



```
{------------------------------------------------}
procedure AXE ( t1 , t2 : integer ) ;
{ traseaza axele de coordonate         }
begin
 moveto ( 0 , -10 )  ; lineto ( 0 , t2 ) ;
 moveto (-5 , t2-5)  ; lineto ( 0 , t2 ) ;
 lineto (+5 , t2-5)  ; moveto (-10, 0 ) ;
 lineto (t1 , 0   )  ; lineto (t1-5,5  ) ;
 moveto (t1-5,-5  )  ; lineto (t1  ,0  ) ;
end;
{------------------------------------------------}
begin { main }
 writeln(' Program pentru trasarea graficului unei functii');
 writeln('     autori : Roxana VLADA   ,   Marin VLADA      ');
 writeln(' Pentru continuare tastati orice tasta ! ');
 ch := readkey ;
 graphdriver := Detect ;
 initgraph(graphdriver, graphmode ,' ') ;{ init mod grafic }
 setViewport ( 50 , 100 , 500 , 300 , false); { originea   }
 setbkcolor ( 1 ) ; setcolor ( 14 ) ;
 a := 0.0  ; b := 40.0 ;
{ se traseaza graficul }
scalX := 10  ; scalY := 45 ; { scalare pe X si Y }
t1 := round( a ) * scalX ; t2 := round ( b ) * scalX ;
moveto ( t1 , round( f( a ) * scalY ) ) ;
for i := t1 + 1 to t2 do
  begin
   val1 := i / 10.0 ;
   lineto ( round( val1*scalX ) , round( f(val1) * scalY ) ) ;
  end;
{ se traseaza axele de coordonate }
AXE ( t2 + 20 , 150 ) ;
ch := readkey ;
closegraph;
end.
```

7.2. Curbe date de ecuații parametrice

În acest caz, curbele plane sînt exprimate de o funcție $F(t) = (f(t), g(t))$, unde $t \in [a, b]$, $F:\mathbf{R} \rightarrow \mathbf{R}\times\mathbf{R}$ (F semnifică un drum în planul $\mathbf{R}\times\mathbf{R}$). Pentru reprezentarea grafică a unei astfel de curbe, se consideră o diviziune a intervalului $[a, b]$ după ce în prealabil s-a realizat o scalare în funcție de cardinalul mulțimii $[a, b] \cap \mathbf{N}$ sau $[a, b] \cap \mathbf{Z}$, după caz. Aceste aspecte practice de implementare pot fi urmărite în programul CURBE care realizează reprezentarea grafică a unor curbe plane remarcabile și anume:

1. Concoida lui NICOMEDE (concoida dreptei)

$$x = a \pm b \cos t, y = a \operatorname{tg} t \pm b \sin t, \quad t \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

2. Melcul lui PASCAL (concoida cercului; cardioida pentru $a = b$)

$$x = 2(a \cos t + b) \cos t$$
$$y = 2(a \cos t + b) \sin t, \quad t \in (-\pi, \pi)$$

3. Cisoida lui DIOCLES

$$x = 2a \sin^2 t$$
$$y = 2a \sin^2 t \cos t, \quad t \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

4. Cisoida elipsei

$$x = \frac{2a^3 \operatorname{tg}^2 t}{a^2 \operatorname{tg}^2 t + b^2}$$
$$y = \frac{2a^3 \operatorname{tg}^3 t}{a^2 \operatorname{tg}^2 t + b^2}, \quad t \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

5. Trisectoarea lui MAC–LAURIN

$$x = a(4\cos^2 t - 1), y = a(4\cos^2 t - 1)\cdot \operatorname{tg} t, \quad t \in (-\frac{\pi}{2}, \frac{\pi}{2})$$

6. Trisectoarea lui LONGCHAMPS

$$x = \frac{a}{4\cos^2 t - 3}, y = \frac{a \operatorname{tg} t}{4\cos^2 t - 3}, \quad t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right) \setminus \left\{\pm\frac{\pi}{6}\right\}$$

7. Cicloida

$$x = at - b \sin t, y = a - b \cos t, \quad t \in \mathbf{R}$$

8. Epicicloida

$$x = (R + r)\cdot \cos \frac{r}{R} t - r \cdot \cos(t + \frac{r}{R} t),$$

$$y = (R + r)\cdot\sin\frac{r}{R}t - r\cdot\sin\left(t + \frac{r}{R}t\right), \quad t \in [0, 2\pi]$$

Observații:

- *dacă R =r se obține cardioida;*
- *forma epicicloidei depinde de raportul r/R;*
- *epicicloida a fost introdusă în anul 1525 de* **Albrecht Dürer** *pentru obținerea de figuri ornamentale.*

9. Hipocicloida

$$x = (R - r)\cos\frac{r}{R}t + r\cos\left(t - \frac{r}{R}t\right),$$

$$y = (R - r)\sin\frac{r}{R}t - r\sin\left(t - \frac{r}{R}t\right), \quad t \in [0, 2\pi];$$

Observație: *pentru R = 4r se va obține astroida.*

10. Astroida

$$x = a\cos^3 t, y = a\sin^3 t, \quad t \in [0, 2\pi]$$

Observație: *Astroida a fost introdusă în anul 1715 de către* **G. Leibnitz.**

11. Strofoida

$$x = a\,(1 \pm \sin t), y = a\,(1 \pm \sin t)\,\mathrm{tg}\,t, \quad t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

12. Bucla Maria Agnesi

$$x = a\,\mathrm{ctg}\,t, y = a\sin^2 t, \quad t \in (0, \pi)$$

Programul CURBE

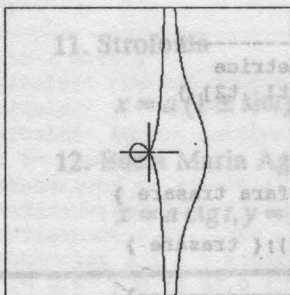
```
{========================================================}
program CURBE ;
uses
  graph , crt ;
var
  graphdriver , graphmode         : integer;
  ch                              : char;
  graphX , graphY                 : array[-319..319] of integer;
  a , b , arg , val1 , val2       : real;
  i , t1 , t2 , flag              : integer;
{========================================================}
procedure INIT ;
{------------}
begin
  graphdriver := Detect;
```

```
    initgraph ( graphdriver , graphmode , ' ' ); { init mod grafic}
    setViewport ( 320 , 175 , 500 , 200 , false );{ fixare origine}
    setbkcolor ( 1 ) ;  { culoare fond  = albastru }
    setcolor ( 14 ) ;   { culoare desen = galben   }
   rectangle ( -150 , -150 , 150 , 150 ); { deseneaza un chenar }
    setViewport ( 170 , 25 , 470 , 325 , true ); { fixare fereastra }
end;
procedure STOP;
{-------------}
{ iesire din modul grafic }
begin
 ch := readkey;   { inghetare imagine }
    closegraph;      { iesire mod grafic }
end;

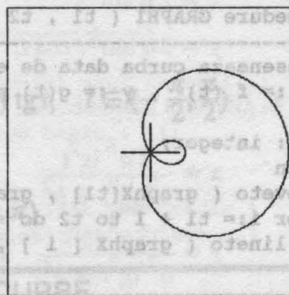
procedure AXE;
{-----------}
{ deseneaza axele (format mic) in origine }
begin
  moveto ( 120 ,  150 );
  lineto ( 180 ,  150 );
  moveto ( 150 ,  120 );
  lineto ( 150 ,  180 );
end;

procedure GRAPH1 ( t1 , t2 : integer ) ;
{-------------------------------------------------}
{ deseneaza curba data de ecuatiile parametrice
  x := f (t)   , y := g(t) pe intervalul [t1 ,t2] }
var
  i : integer;
begin
  moveto ( graphX[t1] , graphY[t1] ) ; { fara trasare }
  for i:= t1 + 1 to t2 do
    lineto ( graphX [ i ] , graphY [ i ] );{ trasare }
end;
{-----------------------------------------------------}
begin { main }
writeln('******************************************************');
writeln('           CURBE PLANE REMARCABILE ');
writeln('               autor : M . Vlada      ');
writeln('******************************************************');
writeln(' 1 =concoida NICOMEDE        2 =melcul lui PASCAL      ');
writeln(' 3 =cisoida DIOCLES          4 =cisoida elipsei        ');
writeln(' 5 =trisectoarea MAC-LAURIN  6 =trisectoarea LONGCHAMPS');
writeln(' 7 =cicloida                 8 =epicicloida            ');
writeln(' 9 =hipocicloida             10=astroida               ');
writeln(' 11=strofoida                12=bucla MARIA AGNESI     ');
writeln(' ( centrul ecranului = originea sistemului cartezian ) ');
writeln('******************************************************');
write(' precizitati numarul de ordine pentru curba dorita : ');
read ( flag ) ;
case flag of
```

```
{ ================  curba lui NICOMEDE ===============}
1 :begin
   writeln(' curba lui NICOMEDE ') ;
   write  (' dati parametri a , b = '); read( a , b ) ;
   t1 := round( (- pi / 2.0)*100) ; t2 :=round( (pi / 2.0)*100);
   { factorul 100 reprezinta scalarea imaginii }
   for i:= t1 to t2 do
     begin
     arg := i / 100.00 ;
     { vectorii graphX, graphY retin coordonatele pentru desenare }
     graphX [i] := round( ( a + b * cos (arg) ) * 100 )+150 ;
     graphY [i] := round((a*sin(arg)/cos(arg) + b*sin(arg))*100)+150;
     end;
   INIT ;
   AXE ;
   GRAPH1 ( t1 , t2 );
   for i:= t1 to t2 do
     begin
     arg := i / 100.00 ;
     graphX[i] := round( ( a - b * cos(arg) ) *100 )+150 ;
     graphY[i] := round( ( a*sin(arg)/cos(arg) - b*sin(arg))*100)+150;
     end;
   GRAPH1 ( t1 , t2 ) ;
   end;
```



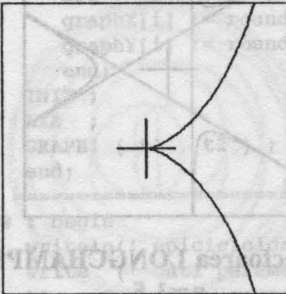
Concoida NICOMEDE
a=0.2, b=0.4



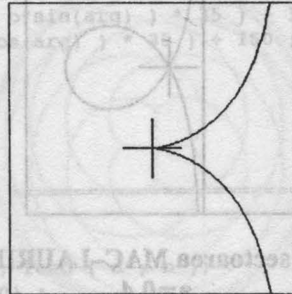
Melcul lui PASCAL
a=0.5, b=0.3

```
{ ==================== melcul lui PASCAL ===================}
2 : begin
   writeln(' melcul lui PASCAL ');
   write  (' dati parametri a ,b = '); read ( a , b ) ;
   t1 := round( -pi * 90 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 90.00 ;
     graphX[i] := round( ( 2 *(a * cos(arg) + b)*cos(arg) ) * 90 );
     graphY[i] := round( ( 2 *(a * cos(arg) + b)*sin(arg) ) * 90 );
     graphX[i] := graphX[i] + 150 ; graphY[i] := graphY[i] + 150  ;
     end;
   INIT ;
```

```
   AXE ;
   GRAPH1 ( t1 , t2 ) ;
   end;
 { =========== cisoida lui DIOCLES ( cisoida cercului =========}
 3 : begin
   writeln(' cisoida lui DIOCLES ( cisoida cercului ');
   write  (' dati parametrul a = ') ; read ( a ) ;
   t1 := round ( (-pi /2.0 ) * 90 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 90.0 ;
     graphX[i] := round( ( 2*a*sin(arg) *sin(arg) ) * 90 ) + 150 ;
     graphY[i] := round( ( 2*a*sin(arg)*sin(arg)*sin(arg) /cos(arg) )
                         * 90 ) + 150 ;
     end;
   INIT ;
   AXE ;
   GRAPH1 ( t1 , t2 ) ;
   end;
```



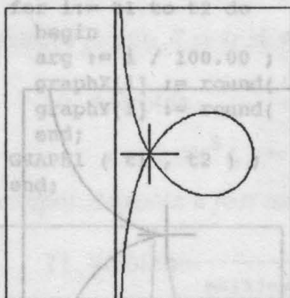
Cisoida lui DIOCLES
a=1



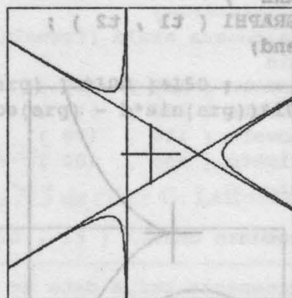
Cisoida elipsei
a=0.8, b=0.4

```
   { ========================= cisoida elipsei ====================}
 4 : begin
   writeln(' cisoida elipsei ') ;
   write  (' dati parametri a , b = ' ); read ( a , b ) ;
   t1 := round( ( - pi / 2.0 ) * 90 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 90.0 + 0.01 ;
     graphX[i] := round( ( 2*a*a*a / ( a*a + b*b* sqr ( cos(arg) /
                         sin(arg) )  ) ) * 90 ) + 150 ;
     graphY[i] := round( ( 2*a*a*a / ( a*a * cos(arg) / sin(arg) +
                         b*b * sqr ( cos(arg) / sin(arg) ) *
                         ( cos(arg) / sin(arg) ) ) ) * 90 ) + 150 ;
     end;
   INIT ;
   AXE ;
   GRAPH1 ( t1 , t2 ) ;
   end;
```

```
{ ================= trisectoarea lui MAC - LAURIN =================}
 5 : begin
   writeln(' trisectoarea lui MAC - LAURIN ' ) ;
   write  (' dati parametrul a = ') ; read ( a ) ;
   t1 := round ( ( - pi / 2.0 ) * 90 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 90.0 + 0.01 ;
     graphX[i] := round (( 4 * a * cos(arg) * cos(arg) - a) * 90)+150;
     graphY[i] := round ( ((4 * a * cos(arg) * cos(arg) - a) *
                      sin(arg) / cos(arg)  ) * 90 ) + 150 ;
   end;
   INIT ;
   AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   end;
```

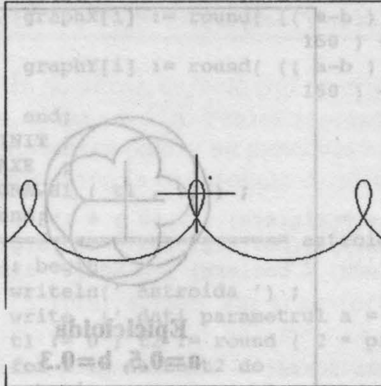


Trisectoarea MAC–LAURIN
a=0.4

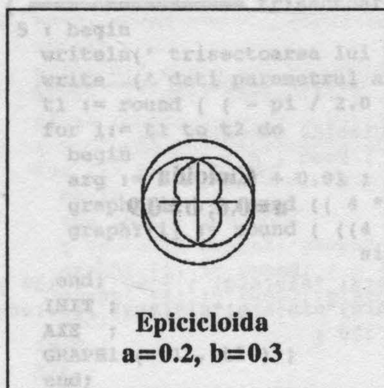


Trisectoarea LONGCHAMPS
a=1.5

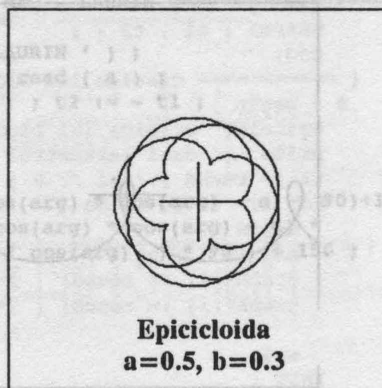
```
{ ================= trisectoarea lui LONGCHAMPS =================}
 6 : begin
   writeln(' trisectoarea lui LONGCHAMPS ') ;
   write  (' dati parametrul a = ') ; read ( a ) ;
   t1 := round ( ( - pi / 2.0 ) * 50 ) ; t2 := - t1 ;
   for  i := t1 to t2 do
     begin
     arg := i / 50.0 + 0.01 ;
     graphX[i] := round ( (a / (4 * cos(arg)*cos(arg) -3) ) *50) +150;
     graphY[i] := round ( ( ( a* sin(arg) /cos(arg) ) / ( 4 *
                      cos(arg) * cos(arg) - 3 ) ) * 50 ) + 150 ;
     end;
   INIT ;
   AXE ;
   GRAPH1 ( t1 , t2 ) ;
   end;
 { ===================== cicloida ============================}
 7 : begin
    writeln(' cicloida ') ;
    write  (' dati parametri a , b = ') ; read( a , b ) ;
    t1 := round ( ( - 2 * pi* 1.4  )* 35 ) ; t2 := - t1 ;
```



```
for i := t1 to t2 do
  begin
  arg := i / 35.0 ;
  graphX[i] := round ( ( a * arg - b*sin(arg) ) * 35 ) + 150 ;
  graphY[i] := round ( ( a - b * cos(arg) ) * 35 ) + 150 ;
  end;
INIT ;
AXE ;
GRAPH1 ( t1 , t2 ) ;
end;
{ ======================== epicicloida ============================}
8 : begin
writeln(' epicicloida ');
write  (' dati parametri a , b =') ; read ( a , b );
t1 := 0 ; t2 := round ( 10 * pi * 10) ;
for i:= t1 to t2 do
  begin
  arg := i / 10.00 ;
  val1 := b *arg / a  ; val2 :=  arg + b * arg / a;
  graphX[i] := round( ( ( a + b) * cos( val1 ) - b *
                       cos( val2  ) ) * 60 ) + 150;
  graphY[i] := round( ( (a + b) * sin( val1 ) - b *
                       sin( val2 ) ) * 60 ) + 150 ;
  end;
INIT ;
AXE  ;
GRAPH1 ( t1 , t2 );
end;
{ ======================== hipocicloida ============================}
9 : begin
writeln(' hipocicloida ') ;
write  (' dati parametri a , b = ') ; read ( a , b ) ;
t1 := 0 ; t2 := round ( 10 * pi * 10 ) ;
for i := t1 to t2 do
  begin
  arg := i / 10.00 ;
  val1 := b * arg / a ; val2 := arg - b * arg / a ;
```

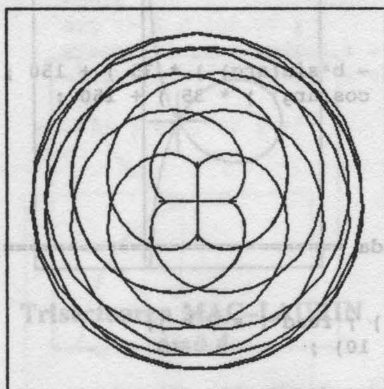


Epicicloida
a=0.2, b=0.3

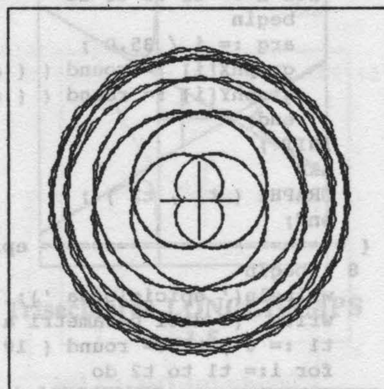


Epicicloida
a=0.5, b=0.3

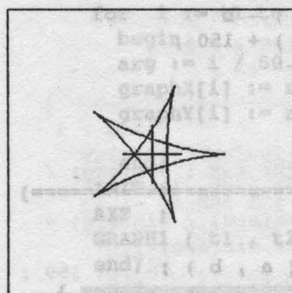
Figuri ornamentale introduse de Dürer, 1525



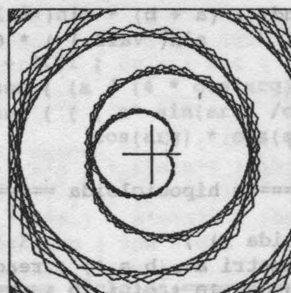
Epicicloida
a=0.4, b=0.9



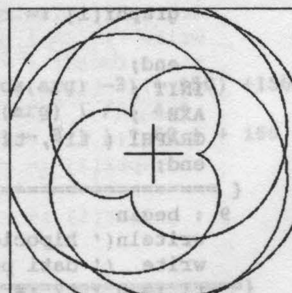
Epicicloida
a=0.2, b=0.9



Hipocicloida
a=0.5, b=0.2



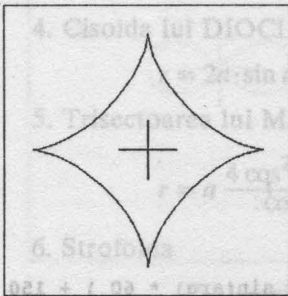
Hipocicloida
a=0.1, b=0.9



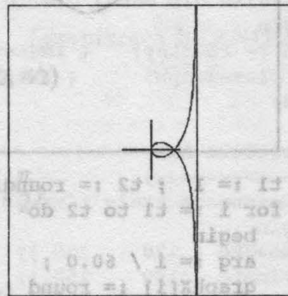
Hipocicloida
a=0.3, b=0.7

```
    graphX[i] := round( (( a-b ) * cos( val1 ) + b * cos( val2 )) *
                        150 ) + 150 ;
    graphY[i] := round( (( a-b ) * sin( val1 ) - b * sin( val2 )) *
                        150 ) + 150 ;
  end;
  INIT ;
  AXE  ;
  GRAPH1 ( t1 , t2 ) ;
  end;
{ ======================== astroida ============================}
10 : begin
  writeln(' astroida ') ;
  write  (' dati parametrul a = '); read ( a ) ;
  t1 := 0 ; t2 := round ( 2 * pi * 10 ) ;
  for i := t1 to t2 do
    begin
    arg := i / 10.00 ;
    graphX[i] := round ( ( a * cos(arg) * cos(arg) * cos(arg) ) *
                        60 ) + 150 ;
    graphY[i] := round ( ( a * sin(arg) * sin(arg) * sin(arg) ) *
                        60 ) + 150 ;
    end;

  INIT ;
  AXE  ;
  GRAPH1 ( t1 , t2 ) ;
  end;
```



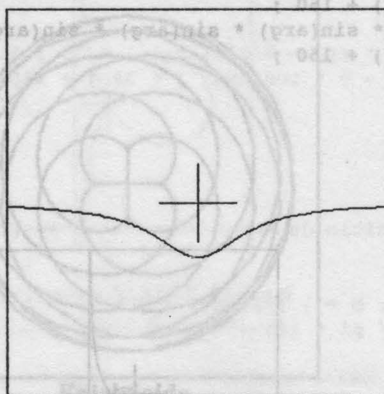
Astroida (G. Leibnitz, 1715)
a=2



Strofoida
a=0.4

```
{ ======================== strofoida ============================}
11 : begin
  writeln(' strofoida ') ;
  write  (' dati parametrul a = '); read ( a ) ;
  t1 := round ( ( - pi / 2 ) * 60 ) ; t2 := -t1 ;
  for i:= t1 to t2 do
    begin
    arg := i / 60.0 + 0.01 ;
    graphX[i] := round( ( a + a * sin(arg) ) * 60 ) + 150 ;
    graphY[i] := round( ( a*sin(arg)/cos(arg) + a * sin(arg) *
```

```
                              sin(arg) / cos(arg) ) * 60 ) + 150 ;
     end;
   INIT ;
   AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   for i := t1 to t2 do
     begin
     arg := i / 60.0 + 0.01 ;
     graphX[i] := round( ( a - a * sin(arg) ) * 60 ) + 150 ;
     graphY[i] := round( ( a*sin(arg)/cos(arg) - a * sin(arg) *
                              sin(arg) / cos(arg) ) * 60 ) + 150 ;
     end;
   GRAPH1 ( t1 , t2 ) ;
   end;
{ =================== bucla MARIA AGNESI ====================}
 12 : begin
   writeln(' bucla MARIA AGNESI ') ;
   write  (' dati parametrul a = ') ; read ( a ) ;
```



Bucla MARIA AGNESI
a=0.7

```
   t1 := 1  ; t2 := round( pi   60 ) - 1 ;
   for i := t1 to t2 do
     begin
     arg := i / 60.0 ;
     graphX[i] := round ( a * cos(arg) / sin(arg) * 60 ) + 150 ;
     graphY[i] := round ( a * sin(arg) * sin(arg) * 60 ) + 150 ;
     end;
   INIT ;
   AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   end;
end;
STOP;
end.
{ ====================  sfirsit program =====================}
```

7.3. Curbe date de ecuaţii polare

În acest caz, curbele sînt exprimate de coordonatele polare (r, t), unde $t \in [a, b]$ iar $r = f(t)$. Pentru reprezentarea grafică a unei astfel de curbe, este necesar ca pentru un punct dat în coordonate polare (r, t) să se realizeze transformarea în coordonate carteziene. Această transformare este:

$$x = f(t)\cos t$$
$$y = f(t)\sin t.$$

În felul acesta am transformat problema pentru curbe plane exprimate cu ecuaţii parametrice. Vom prezenta în limbajul **TURBO PASCAL** programul POLARE care realizează reprezentarea grafică a unor curbe plane remarcabile, şi anume:

1. Lemniscata lui BERNOULLI

$$r = \pm a\sqrt{2\cos 2t}, \quad t \in (-\pi/4, \pi/4)$$

2. Concoida lui NICOMEDE

$$r = \frac{a}{\cos t} \pm b, \quad t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

3. Melcul lui PASCAL

$$r = 2a\cos t \pm 2b, \quad t \in [-\pi/2, \pi/2]$$

4. Cisoida lui DIOCLES

$$r = 2a \cdot \sin t \cdot \operatorname{tg} t, \quad t \in (-\pi/2, \pi/2)$$

5. Trisectoarea lui MAC–LAURIN

$$r = a\,\frac{4\cos^2 t - 1}{\cos t}, \quad t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

6. Strofoida

$$r = a\,\frac{1 \pm \sin t}{\cos t}, \quad t \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

7. Spirala logaritmică

$$r = a \cdot e^{1+t}, \quad t \in (0, \infty)$$

8. Spirala hiperbolică

$$r = \frac{a}{t}, \quad t \in (0, \infty)$$

9. Spirala lui ARHIMEDE

$$r = a\,t, \quad t \in (0, \infty)$$

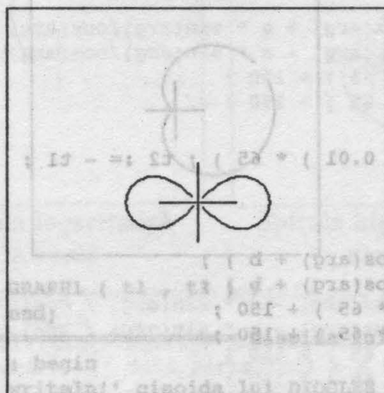
Programul POLARE

```
{===========================================================}
program POLARE ;  { autor: M. Vlada }
uses
  graph , crt ;
var
  graphdriver , graphmode    : integer;
  ch                         : char;
  graphX , graphY                   : array[-319..319] of integer;
  a , b , arg , val1 ,val2          : real;
  i , t1 , t2 , flag                : integer;
{===========================================================}
procedure INIT ;
{-------------}
begin
  graphdriver := Detect;
   initgraph ( graphdriver , graphmode , ' ' ); { init mod grafic}
   setViewport ( 320 , 175 , 500 , 200 , false );{ fixare origine}
   setbkcolor ( 1 ) ;  { culoare fond  = albastru }
   setcolor ( 14 ) ;   { culoare desen = galben    }
   rectangle ( -150 , -150 , 150 , 150 ); { deseneaza un chenar }
   setViewport ( 170 , 25 , 470 , 325 , true ); { fixare fereastra }
end;
procedure STOP;
{-------------}
{ iesire din modul grafic }
begin
 ch := readkey;    { inghetare imagine }
   closegraph;      { iesire mod grafic }
end;

procedure AXE;
{-----------}
{ deseneaza axele (format mic) in origine }
begin
  moveto ( 120 ,  150 );
  lineto ( 180 ,  150 );
  moveto ( 150 ,  120 );
  lineto ( 150 ,  180 );
end;

procedure GRAPH1 ( t1 , t2 : integer ) ;
{-------------------------------------------------------}
{ deseneaza curba data de ecuatia polara
  r =  f (t)  pe intervalul [t1 ,t2]       }
var
  i : integer;
begin
  moveto ( graphX[t1] , graphY[t1] ) ; { fara trasare }
  for i:= t1 + 1 to t2 do
```

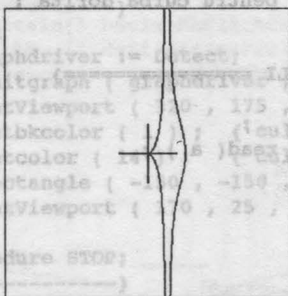
```
    lineto ( graphX [ i ] , graphY [ i ] );{ trasare }
end;
{-----------------------------------------------------}
begin { main }
writeln('*******************************************************');
writeln('          CURBE PLANE REMARCABILE ');
writeln('              autor : M . Vlada     ');
writeln('*******************************************************');
writeln(' 1 =lemniscata BERNOULLI    2 =concoida NICOMEDE    ');
writeln(' 3 =melcul lui PASCAL       4 =cisoida lui DIOCLES  ');
writeln(' 5 =trisectoarea MAC-LAURIN 6 =strofoida            ');
writeln(' 7 =spirala logaritmica     8 =spirala hiperbolica  ');
writeln(' 9 =spirala lui ARHIMEDE                            ');
writeln(' ( centrul ecranului = originea sistemului cartezian ) ');
writeln('*******************************************************');
write(' precizitati numarul de ordine pentru curba dorita : ');
read ( flag ) ;
case flag of
{  =============== lemniscata BERNOULLI ===============}
 1 : begin
    writeln(' lemniscata BERNOULLI ')    ;
    write  (' dati parametrul a  = '); read( a ) ;
```



Lemniscata BERNOULLI
a=0.8

```
    t1 := round( (- pi / 4.0)*50) ; t2 :=round( (pi / 4.0)*50);
    { factorul 65 reprezinta scalarea imaginii }
    for i:= t1 to t2 do
      begin
      arg := i / 50.00 ;
      { vectorii graphX, graphY retin coordonatele pentru desenare }
      val1 := cos(arg) * a * sqrt( 2.0 * cos( 2.0 * arg) ) ;
      val2 := sin(arg) * a * sqrt( 2.0 * cos( 2.0 * arg) ) ;
      graphX [i] := round(  val1  * 50 ) + 150 ;
      graphY [i] := round(  val2  * 50 ) + 150 ;
      end;
    INIT ;
    AXE ;
    GRAPH1 ( t1 , t2 );
    t1 := round ( ( 3.0* pi / 4.0) * 50 ) ;
```

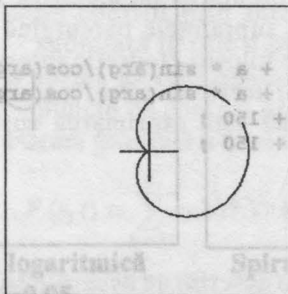
```
    t2 := round ( ( 5.0 * pi / 4.0 ) * 50 ) ;
    for i:= t1 to t2 do
      begin
      arg := i / 50.00 ;
      val1 :=  cos(arg) * a * sqrt( 2.0 * cos( 2.0 * arg) ) ;
      val2 :=  sin(arg) * a * sqrt( 2.0 * cos( 2.0 * arg) ) ;
      graphX[i] := round ( val1 * 50 ) + 150  ;
      graphY[i] := round ( val2 * 50 ) + 150 ;
      end;
    GRAPH1 ( t1 , t2 ) ;
    end;
{ ================  concoida NICOMEDE     =============}
 2 : begin
    writeln(' concoida lui NICOMEDE ' ) ;
    write  (' dati parametri a , b = ' ) ; read ( a , b ) ;
```



Concoida lui NICOMEDE
a=b=0.3

```
    t1 := round ( ( - pi / 2.0 + 0.01 ) * 65 ) ; t2 := - t1 ;
    for i := t1 to t2 do
      begin
      arg := i / 65 ;
      val1 := cos(arg) * ( a / cos(arg) + b ) ;
      val2 := sin(arg) * ( a / cos(arg) + b ) ;
      graphX[i] := round ( val1 * 65 ) + 150 ;
      graphY[i] := round ( val2 * 65 ) + 150 ;
      end;
   INIT ;
   AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   for i:= t1 to t2 do
    begin
      arg := i / 65.00 ;
      val1 := cos(arg) * ( a / cos(arg) - b ) ;
      val2 := sin(arg) * ( a / cos(arg) - b ) ;
      graphX[i] := round ( val1 * 65 ) + 150 ;
      graphY[i] := round ( val2 * 65 ) + 150 ;
      end;
   GRAPH1 (t1 , t2 ) ;
   end;
{=================== melcul lui PASCAL ================}
 3 : begin
    writeln(' melcul lui PASCAL ' ) ;
    write  (' dat parametri a , b = ' ) ; read ( a , b ) ;
```

```
   t1 := round ( ( - pi / 2.0 ) * 65 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 65.00 ;
     val1 := cos(arg) * ( 2 * a * cos(arg) + 2 * b ) ;
     val2 := sin(arg) * ( 2 * a * cos(arg) + 2 * b ) ;
     graphX[i] := round ( val1 * 65 ) + 150 ;
     graphY[i] := round ( val2 * 65 ) + 150 ;
     end;
     INIT ;
     AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   for  i:= t1 to t2 do
     begin
     arg := i / 65.00 ;
     val1 := cos(arg) * ( 2 * a * cos(arg) - 2 * b ) ;
     val2 := sin(arg) * ( 2 * a * cos(arg) - 2 * b ) ;
     graphX[i] := round ( val1 * 65 ) + 150;
     graphY[i] := round ( val2 * 65 ) + 150;
     end;
```

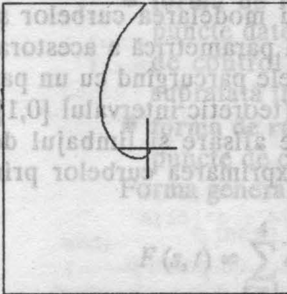


Melcul lui PASCAL (cardioida)
a=b=0.4

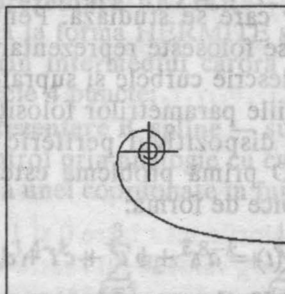
```
   GRAPH1 ( t1 , t2 ) ;
   end;
 {==================== cisoida lui DIOCLES ===================}
 4 : begin
   writeln(' cisoida lui DIOCLES ' ) ;
   write  (' dati parametrul a = ' ) ; read ( a ) ;
   t1 := round ( ( - pi / 2 ) * 65 ) ; t2 := - t1 ;
   for i:= t1 to t2 do
     begin
     arg := i / 65.00 ;
     val1 := cos(arg) * ( 2 * a * sin(arg) * sin(arg) / cos(arg) );
     val2 := sin(arg) * ( 2 * a * sin(arg) * sin(arg) / cos(arg) );
     graphX[i] := round( val1 * 65 ) + 150 ;
     graphY[i] := round( val2 * 65 ) + 150 ;
     end;
   INIT ;
   AXE  ;
   GRAPH1 ( t1 , t2 ) ;
   end;
 {=================== trisectoarea MAC - LAURIN ===============}
 5 : begin
```

```
    writeln(' trisectoarea MAC - LAURIN ') ;
    write  (' dati parametrul a = ') ; read ( a ) ;
    t1 := round ( ( - pi / 2 ) * 65 ) ; t2 := - t1 ;
    for i := t1 to t2 do
      begin
      arg := i / 65.00 ;
      val1 := cos(arg) * ( a * ( 4*cos(arg) - 1/cos(arg) ) ) ;
      val2 := sin(arg) * ( a * ( 4*cos(arg) - 1/cos(arg) ) ) ;
      graphX[i] := round ( val1 * 65 ) + 150 ;
      graphY[i] := round ( val2 * 65 ) + 150 ;
      end;
    INIT ;
    AXE  ;
    GRAPH1 ( t1 , t2 ) ;
    end;
 {===================== strofoida ===========================}
  6 : begin
    writeln(' strofoida ');
    write  (' dati parametrul a = ') ; read ( a ) ;
    t1 := round( ( - pi / 2 ) * 65 ) ; t2 := - t1 ;
    for i:= t1 to t2 do
      begin
      arg := i / 65.00 ;
      val1 := cos(arg) * ( a / cos(arg) + a * sin(arg)/cos(arg) );
      val2 := sin(arg) * ( a / cos(arg) + a * sin(arg)/cos(arg) );
      graphX[i] := round ( val1 * 65 ) + 150 ;
      graphY[i] := round ( val2 * 65 ) + 150 ;
      end;
    INIT ;
    AXE  ;
    GRAPH1 ( t1 , t2 ) ;
    for i:= t1 to t2 do
      begin
      arg := i / 65.00 ;
      val1 := cos (arg) * ( a / cos(arg) - a * sin(arg) / cos(arg) );
      val2 := sin (arg) * ( a / cos(arg) - a * sin(arg) / cos(arg) );
      graphX[i] := round ( val1 * 65 ) + 150 ;
      graphY[i] := round ( val2 * 65 ) + 150 ;
      end;
    GRAPH1 ( t1 , t2 ) ;
    end;
 {===================== spirala logaritmica ====================}
  7 : begin
    writeln(' spirala logaritmica ' ) ;
    write  (' dati parametrul a = ')  ; read ( a ) ;
    t1 := 1    ; t2 := 15 * 10;
    for i:= t1 to t2 do
      begin
      arg := i / 10.00 ;
      val1 := cos(arg) * a * exp ( 1.00 + arg ) ;
      val2 := sin(arg) * a * exp ( 1.00 + arg ) ;
      graphX[i] := round ( val1 * 10 ) + 150    ;
      graphY[i] := round ( val2 * 10 ) + 150    ;
```

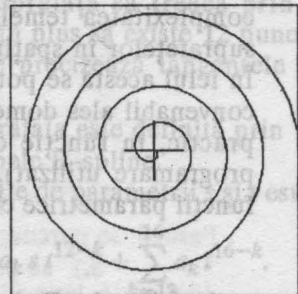
```
        end;
      INIT ;  AXE   ;  GRAPH1 ( t1 , t2 ) ;
      end;
    {================= spirala hiperbolica ===================}
     8 : begin
      writeln(' spirala hiperbolica ') ;
      write   (' dati parametrul a = ') ; read ( a ) ;
      t1 := 1   ; t2 := 15 * 10 ;
      for i:= t1 to t2 do
        begin
        arg := i / 10.00 ;
        val1 := cos(arg) * a / arg ; val2 := sin(arg) * a / arg ;
        graphX[i] := round ( val1 * 10) + 150 ;
        graphY[i] := round ( val2 * 10 ) + 150 ;
        end;
      INIT ;
      AXE  ;
      GRAPH1 ( t1 , t2 ) ;
      end;
```



Spirala logaritmică
a=0.05



Spirala hiperbolică
a=7



Spirala lui ARHIMEDE
a=0.6

```
    {================== spirala lui ARHIMEDE ====================}
     9 : begin
      writeln(' spirala lui ARHIMEDE ' ) ;
      write   (' dati parametrul a = ') ; read ( a ) ;
      t1 := 1   ; t2 := 25 * 10 ;
      for i := t1 to t2 do
        begin
        arg := i / 10.00 ;
        val1 := cos(arg) * a * arg ; val2 := sin(arg) * a * arg ;
        graphX[i] := round ( val1 * 10 ) + 150 ;
        graphY[i] := round ( val2 * 10 ) + 150 ;
        end;
      INIT ;
      AXE  ;
      GRAPH1 ( t1 , t2 ) ;
      end;
    end;
    STOP;
    end.
    { ====================  sfirsit program ====================}
```

7.4. Curbe şi suprafeţe în spaţiu

În domeniul aplicaţiilor grafice pe calculator, o importanţă deosebită o are modelarea corpurilor în spaţiu, precum şi studiul imaginilor provenite din analiza unor procese. Reprezentarea imaginilor pe ecranul unui dispozitiv grafic se face în mai multe moduri astfel încît aceasta să fie cît mai sugestivă:

- reprezentări prin puncte (reprezentări prin secţiuni transversale)
- reprezentări tip „wire-frame" („cadru de sîrmă")
- reprezentarea prin reţea de poligoane (reprezentarea poliedrală)

Toate aceste reprezentări ridică fiecare probleme specifice, în literatura de specialitate acestea fiind tratate cu mare atenţie în funcţie de aplicaţiile grafice în care se întîlnesc.

Curbele şi suprafeţele în spaţiu sînt mai dificil de reprezentat, dar necesitatea lor în aplicaţiile grafice este evidentă dacă se ţine seama de complexitatea temelor care se studiază. Pentru modelarea curbelor şi suprafeţelor în spaţiu se foloseşte reprezentarea parametrică a acestora. În felul acesta se pot descrie curbele şi suprafeţele parcurgînd cu un pas convenabil ales domeniile parametrilor folosiţi (teoretic intervalul [0,1], practic, în funcţie de dispozitivul periferic de afişare şi limbajul de programare utilizat). O primă problemă este exprimarea curbelor prin funcţii parametrice cubice de forma:

$$F(t) = a\,t^3 + b\,t^2 + c\,t + d,$$

unde $a, b, c, d \in \mathbf{R}$ şi $t \in [0, 1]$.

În felul acesta o curbă în spaţiu este descrisă de ecuaţiile parametrice:

$$x = x(t)$$
$$y = y(t)$$
$$z = z(t)$$

unde $x(t)$, $y(t)$, $z(t)$ sînt funcţii parametrice cubice care se obţin din expresia lui F prin înlocuirea parametrilor a, b, c, d cu coeficienţii corespunzători. Pentru reprezentarea imaginii curbei pe ecran există mai multe modalităţi:

- **forma de reprezentare HERMITE** — capetele curbei să coincidă cu două puncte date şi tangentele la capetele curbei să coincidă cu două direcţii date;
- **forma de reprezentare BÉZIER** — capetele curbei să coincidă cu două puncte date şi tangentele la capetele curbei să coincidă cu două direcţii determinate de capetele curbei şi două puncte de control alese corespunzător;
- **forma de reprezentare B-spline** — se foloseşte o mulţime de puncte de control.

Pentru reprezentarea suprefeţelor în spaţiu vom folosi două familii de curbe cubice corespunzătoare celor două direcţii într-un plan xOy. Prin urmare, ecuaţia suprafeţei trebuie să se exprime în funcţie de doi parametrii s şi t, adică:

$$\begin{aligned} x &= x\,(t,s) \\ y &= y\,(t,s) \\ z &= z\,(t,s), \quad \text{unde } s,t \in [0,1]. \end{aligned}$$

Aceste relaţii sînt ecuaţiile unei suprafeţe **bicubice**. Pentru reprezentarea imaginii suprafeţei pe ecran există şi în acest caz trei modalităţi:

- **forma de reprezentare HERMITE** — suprafaţa să treacă prin patru puncte din spaţiu, corespunzătoare valorilor extreme 0 şi 1 pentru parametrii s şi t (acestea se notează cu P_{00}, P_{01}, P_{10}, P_{11}) şi să aibă trei tangente la suprafaţă date în fiecare din aceste puncte;
- **forma de reprezentare BÉZIER** — suprafaţa să treacă prin 4 puncte date ca la forma HERMITE şi în plus să existe 12 puncte de control prin intermediul cărora se precizează tangentele la suprafaţă în cele 4 puncte;
- **forma de reprezentare B-spline** — suprafaţa este definită prin 16 puncte de control prin analogie cu curbele B-spline.

Forma generală a unei coordonate în funcţie de parametrii s şi t este:

$$F(s,t) = \sum_{k=1}^{4} a_k s^3 t^{4-k} + \sum_{k=5}^{8} a_k s^2 t^{8-k} + \sum_{k=9}^{12} a_k s\, t^{12-k} + \sum_{k=13}^{16} a_k t^{16-k}.$$

În continuare, vom prezenta elementele necesare pentru reprezentarea suprafeţelor sub formă HERMITE prin intermediul unui algoritm implementat ulterior într-un program scris în limbajul **TURBO PASCAL** pentru microcalculatoare compatibile IBM-PC.

Fie P_{ij}, $i,j \in \{0,1\}$ cele 4 puncte de control corespunzătoare valorilor extreme 0 şi 1 ale parametrilor s şi t prin care trebuie să treacă suprafaţa, adică $P_{ij}\,(x_{ij}, y_{ij}, z_{ij})$, $i,j \in \{0,1\}$ sînt puncte cu coordonatele precizate ca date de intrare.

Vom nota prin $\frac{\partial F}{\partial s}, \frac{\partial F}{\partial t}, \frac{\partial^2 F}{\partial s\,\partial t}$ derivatele de ordinul 1 şi 2 pentru funcţia F.

Fie

$$T_{sij} = \left(\frac{\partial x}{\partial s}\bigg|_{(i,j)}, \frac{\partial y}{\partial s}\bigg|_{(i,j)}, \frac{\partial z}{\partial s}\bigg|_{(i,j)} \right), \quad i,j \in \{0,1\}$$

$$T_{tij} = \left(\frac{\partial x}{\partial t}\bigg|_{(i,j)}, \frac{\partial y}{\partial t}\bigg|_{(i,j)}, \frac{\partial z}{\partial t}\bigg|_{(i,j)} \right), \quad i,j \in \{0,1\}$$

$$T_{stij} = \left(\left.\frac{\partial^2 x}{\partial s\,\partial t}\right|_{(i,j)}, \left.\frac{\partial^2 y}{\partial s\,\partial t}\right|_{(i,j)}, \left.\frac{\partial^2 z}{\partial s\,\partial t}\right|_{(i,j)} \right), \quad i,j \in \{0,1\}$$

parametrii tangentelor în cele 4 puncte de control, unde $x(s,t)$, $y(s,t)$, $z(s,t)$ sînt funcţii de forma F.

Generarea imaginii suprafeţei se face cu ajutorul următorului algoritm:

Algoritm pentru generarea unei suprafețe HERMITE

Pasul 1: se citeşte pasul de parcurgere p.

Pasul 2: se citeşte unghiul axei Oz cu axa Ox, φ.

Pasul 3: se citeşte raza r;
se citesc coordonatele celor 4 puncte de control:

$$(x_{ij}, y_{ij}, z_{ij}), \quad i,j \in \{0,1\}$$

Pasul 4: se citesc parametrii tangentelor la suprafaţă:

$$(a_{sij}, b_{sij}, c_{sij}), \quad i,j \in \{0,1\}$$

$$(a_{tij}, b_{tij}, c_{tij}), \quad i,j \in \{0,1\}$$

$$(a_{stij}, b_{stij}, c_{stij}), \quad i,j \in \{0,1\}.$$

Pasul 5: se parcurg valorile lui s între 0 şi 1 cu pasul p
- 5.1 se parcurg valorile lui t între 0 şi 1 cu pasul p
  - 5.1.1. se calculează coordonatele $x(s,t)$, $y(s,t)$, $z(s,t)$
  - 5.1.2. se calculează proiecţia punctului (x,y,z)
  - 5.1.3. se calculează coordonatele punctului imagine
- 5.2. se trece la următorul t
- 5.3. se trece la următorul s.

Pasul 6: se trasează cele două familii de curbe.

Pasul 7: stop ■

Programul SUPRAFEȚE

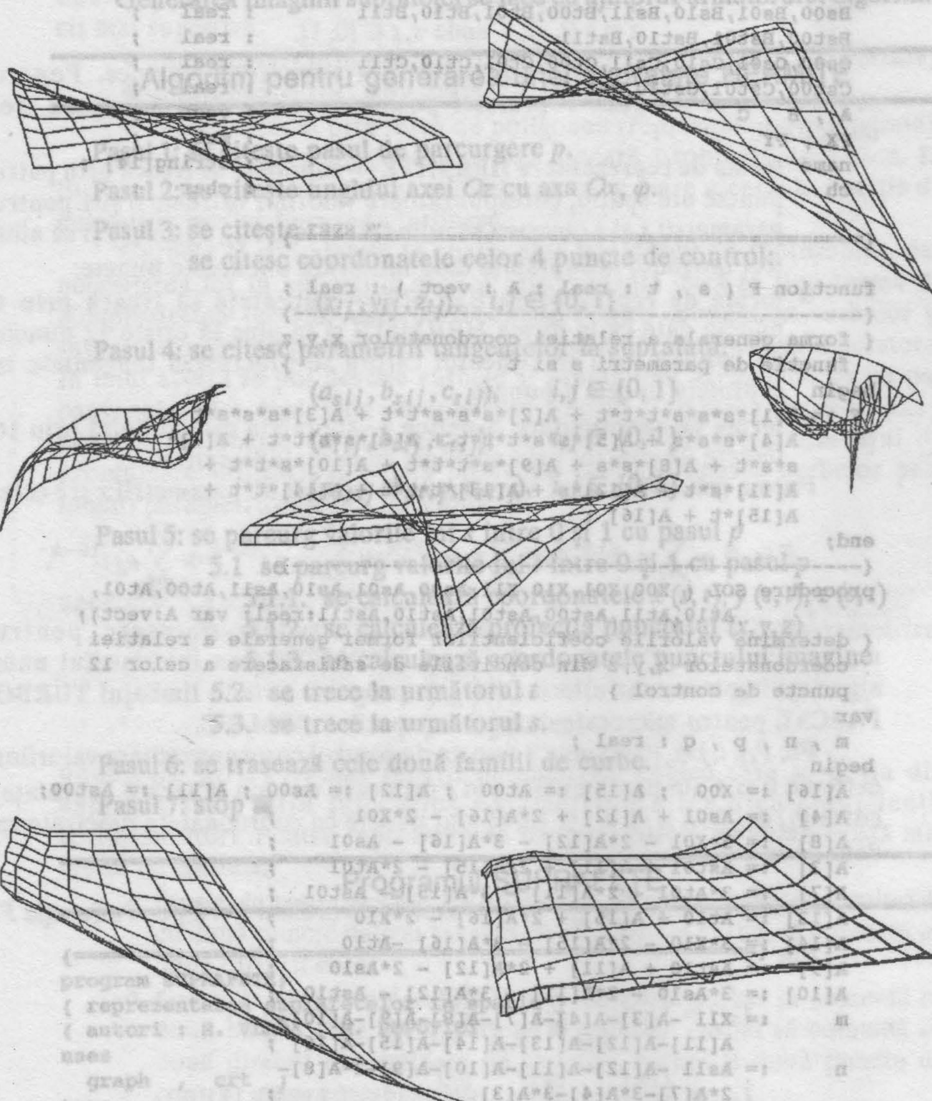
```
{=====================================================}
program SUPRAFETE;
{ reprezentarea suprafetelor in spatiu }
{ autori : M. VLADA , M. POPOVICI      }
uses
  graph  ,  crt  ;
type
  vect = array[1..16] of real ;
  mat  = array[0..50,0..50] of real;
const
```

```
 solid :fillpatterntype=($FF,$FF,$FF,$FF,$FF,$FF,$FF,$FF);
var
  graphdriver , graphmode                          : integer;
  s , t , i , j , sc                               : integer;
  x , y , z , p , fi , r , h , g                   : real    ;
  X00,X01,X10,X11,Y00,Y01,Y10,Y11,Z00,Z01,Z10,Z11: real    ;
  As00,As01,As10,As11,At00,At01,At10,At11          : real    ;
  Ast00,Ast01,Ast10,Ast11                          : real    ;
  Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,Bt10,Bt11          : real    ;
  Bst00,Bst01,Bst10,Bst11                          : real    ;
  Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,Ct10,Ct11          : real    ;
  Cst00,Cst01,Cst10,Cst11                          : real    ;
  A , B , C                                        : vect    ;
  PX , PY                                          : mat     ;
  name                                             : string[14] ;
  ch                                               : char    ;

{==============================================}

function F ( s , t : real ; A : vect ) : real ;
{----------------------------------------------}
{ forma generala a relatiei coordonatelor x,y,z
  functie de parametri s si t                 }
begin
 F := A[1]*s*s*s*t*t*t + A[2]*s*s*s*t*t + A[3]*s*s*s*t +
      A[4]*s*s*s + A[5]*s*s*t*t*t + A[6]*s*s*t*t + A[7]*
      s*s*t + A[8]*s*s + A[9]*s*t*t*t + A[10]*s*t*t +
      A[11]*s*t + A[12]*s + A[13]*t*t*t + A[14]*t*t +
      A[15]*t + A[16] ;
end;
{----------------------------------------------}
procedure SOL ( X00,X01,X10,X11,As00,As01,As10,As11,At00,At01,
          At10,At11,Ast00,Ast01,Ast10,Ast11:real; var A:vect);
{ determina valorile coeficientilor formei generale a relatiei
  coerdonatelor x,y,z din conditiile de satisfacere a celor 12
  puncte de control }
var
  m , n , p , q : real ;
begin
  A[16] := X00  ; A[15] := At00  ; A[12] := As00 ; A[11] := Ast00;
  A[4]  := As01 + A[12] + 2*A[16] - 2*X01      ;
  A[8]  := 3*X01 - 2*A[12] - 3*A[16] - As01    ;
  A[3]  := Ast01 + A[11] + 2*A[15] - 2*At01    ;
  A[7]  := 3*At01 - 2*A[11] - 3*A[15] - Ast01  ;
  A[13] := At10 + A[15] + 2*A[16] - 2*X10      ;
  A[14] := 3*X10 - 2*A[15] - 3*A[16] -At10     ;
  A[9]  := Ast10 + A[11] + 2*A[12] - 2*As10    ;
  A[10] := 3*As10 - 2*A[11] - 3*A[12] - Ast10  ;
  m     := X11 -A[3]-A[4]-A[7]-A[8]-A[9]-A[10]-
           A[11]-A[12]-A[13]-A[14]-A[15]-A[16] ;
  n     := As11 -A[12]-A[11]-A[10]-A[9]-2*A[8]-
           2*A[7]-3*A[4]-3*A[3]                ;
  p     := At11 -A[15]-A[11]-2*A[14]-2*A[10] -
```

Exemple de suprafeţe



```
          3*A[13]-3*A[9]-A[7]-A[3]                ;
  q   := Ast11 -A[11] -2*A[10] -2*A[7] -
          3*A[9]-3*A[3]                            ;
  A[5]  := 3*p - q - 6*m + 2*n                     ;
  A[6]  := 9*m - 3*( n + p ) + q                   ;
  A[1]  := q - 2*( n + p ) + 4*m                   ;
  A[2]  := 3*n + 2*p - q - 6*m                     ;
end;
{-------------------------------------------------}
begin { main }
  writeln('*******************************************');
  writeln('   reprezentarea suprafetelor in spatiu   ');
  writeln('      autori : M. VLADA , M. POPOVICI      ');
  writeln('*******************************************');
  write   ('Datele se citesc dintr-un fisier ? ( D ,N ) : ');
  readln(ch);
  if (ch = 'D') then
    begin
      write(' Nume fisier intrare : ');
      read(name);
      assign(input, name);
      reset(input);
      readln( p , fi , r ) ;
      readln( X00 , X01 , X10 , X11 ) ; { coordonatele }
      readln( Y00 , Y01 , Y10 , Y11 ) ; { x,y,z pentru }
      readln( Z00 , Z01 , Z10 , Z11 ) ; { punctele de }
                                        { de control   }
      { parametrii tangentelor la suprafata             }
      readln( As00,As01,As10,As11,At00,At01,At10,At11 );
      readln( Ast00,Ast01,Ast10,Ast11 ); { pentru x   }
      readln( Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,Bt10,Bt11 );
      readln( Bst00,Bst01,Bst10,Bst11 ); { pentru y    }
      readln( Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,Ct10,Ct11 );
      readln( Cst00,Cst01,Cst10,Cst11 ) ; { pentru z }
    end
  else
    begin
      write(' dati pasul p = ') ; read( p ) ;
      write(' unghiul axei OZ cu OX ( 0..90) ; fi= ');
      read ( fi ) ;
      write(' dati raza r = '); read ( r ) ;
      writeln(' COORDONATELE PUNCTELOR DE CONTROL');
      writeln('    - coordonata x ');
      write(' X00,X01,X10,X11 = ') ; read(X00,X01,X10,X11);
      writeln('    - coordonata y ');
      write(' Y00,Y01,Y10,Y11 = ') ; read(Y00,Y01,Y10,Y11);
      writeln('    - coordonata z ');
      write(' Z00,Z01,Z10,Z11 = ') ; read(Z00,Z01,Z10,Z11);
      writeln('PARAMETRI TANGENTELOR IN PUNCTELE DE CONTROL');
      writeln('    - coordonata x ');
      writeln(' As00,As01,As10,As11,At00,At01,At10,At11,',
              'Ast00,Ast01,Ast10,ASt11 = ');
      read(As00,As01,As10,As11,At00,At01,At10,At11,
```

```
                 Ast00,Ast01,Ast10,Ast11) ;
    writeln('    - coordonata y ');
    writeln(' Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,Bt10,Bt11,',
                 'Bst00,Bst01,Bst10,Bst11 = ');
    read(Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,Bt10,Bt11,
                 Bst00,Bst01,Bst10,Bst11) ;
    writeln('    - coordonata z ');
    writeln(' Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,Ct10,Ct11,',
                 'Cst00,Cst01,Cst10,Cst11 = ');
    read(Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,Ct10,Ct11,
                 Cst00,Cst01,Cst10,Cst11) ;
   { scrierea datelor in fisier }
    readln;
    write(' Doriti memorarea datelor in fisier ?( D,N):');
    readln(ch);
    if ch ='D' then
    begin
    write(' Nume fisier iesire : ');
    read(name);
    assign ( output, name);
    rewrite ( output);
    writeln(p , fi , r ) ;
    writeln(X00,X01,X10,X11) ;
    writeln(Y00,Y01,Y10,Y11) ;
    writeln(Z00,Z01,Z10,Z11) ;
    writeln(As00,As01,As10,As11,At00,At01,At10,At11);
    writeln(Ast00,Ast01,Ast10,Ast11) ;
    writeln(Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,Bt10,Bt11);
    writeln(Bst00,Bst01,Bst10,Bst11) ;
    writeln(Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,Ct10,Ct11);
    writeln(Cst00,Cst01,Cst10,Cst11);
    end;
  end;
{ calculul parametrilor formei generale pentru expresia
  coordonatelor functie de parametri s si t        }
  SOL ( X00,X01,X10,X11,As00,As01,As10,As11,At00,At01,
       At10,At11,Ast00,Ast01,Ast10,Ast11, A ) ;
  SOL ( Y00,Y01,Y10,Y11,Bs00,Bs01,Bs10,Bs11,Bt00,Bt01,
       Bt10,Bt11,Bst00,Bst01,Bst10,Bst11, B ) ;
  SOL ( Z00,Z01,Z10,Z11,Cs00,Cs01,Cs10,Cs11,Ct00,Ct01,
       Ct10,Ct11,Cst00,Cst01,Cst10,Cst11, C ) ;
  h  := cos ( fi ) ; g := sin ( fi ) ;
{ generarea retelei conform variatiei parametrilor s si t }
  i := 0 ;
  for s := 0 to round ( p * 100 ) do
   begin
    j := 0 ;
    for t := 0 to round ( p * 100 ) do
     begin
      x := F ( s/10.00 , t/10.00 , A ) ;
      y := F ( s/10.00 , t/10.00 , B ) ;
      z := F ( s/10.00 , t/10.00 , C ) ;
      PX [ i,j ] := x + z * h * r ;
```

```
      PY [ i,j ] := y + z * g * r ;
      j := j + 1 ;
      end;
    i := i + 1 ;
    end;
{ reprezentarea grafica a suprafetei }
  graphdriver := Detect ;
  initgraph ( graphdriver, graphmode, ' ') ; { init mod grafic }
  setfillpattern(solid,15);
  bar(0,0,700,500) ;
  SETCOLOR(0) ;
  setViewport ( 60,30, 500 , 200 , false); { originea
 { setbkcolor ( 1 ) ;  culoare fond  = albastru }
 { setcolor ( 14 )  ;  culoare desen = galben   }
  sc := 5 ;
  for i := 0 to round ( p * 100 ) do
    for j:= 0 to round ( p * 100 ) - 1 do
      begin
      moveto ( round( PX[i,j] ) * sc , round ( PY[i,j] ) * sc);
      lineto ( round( PX[i,j+1])* sc , round ( PY[i,j+1])* sc) ;
      end;
  for j := 0 to round( p * 100 ) do
    for i := 0 to round ( p * 100 ) - 1 do
      begin
      moveto ( round( PX[i,j] )*sc , round( PY[i,j] )*sc )  ;
      lineto ( round( PX[i+1,j])*sc , round( PY[i+1,j])*sc) ;
      end;
  ch :=readkey;
  closegraph;
  end.
{ ================= sfirsit program ===========================}
```

7.5. Probleme propuse

1) Să se reprezinte grafic funcţiile date de următoarele ecuaţii:

- $f(x) = x\cos\frac{1}{x},\ x \in [-10, 10] \setminus \{0\}$
- $f(x) = \begin{cases} 0, & \text{pentru } x = 0 \\ x^p \sin\frac{1}{x}, & \text{pentru } x \in [-10, 10], x \neq 0 \end{cases}$, unde $p \in \mathbf{N}$ este fixat
- $f(x) = x^{1/x},\ x \in (0, 100)$.

2) Să se reprezinte grafic curbele date de următoarele ecuaţii parametrice:

- $x(t) = \dfrac{a}{1 + a^2 t}$

  $y(t) = \dfrac{a^2 t}{\left(1 + a^2 t^2\right)^2},\quad t \in \mathbf{R}$
- $x(t) = \frac{a}{3}(2\cos t + \cos 2t)$

  $y(t) = \frac{a}{3}(2\cos t - \sin 2t),\quad t \in [0, 2\pi]$

- $x(t) = \dfrac{2t(3t^2-4)}{1+t^2}$

  $y(t) = \dfrac{2t^2(3t^2-4)}{1+t^2}, \quad t \in \mathbf{R}$

3) Să se reprezinte grafic curbele date de următoarele ecuaţii polare:

- $r = a\cos\frac{3t}{4}, \quad t \in [-2\pi, 2\pi]$
- $r = a\sin t, \quad t \in (0, 2\pi), a \in \mathbf{R}$
- $r = \sin(nt)\cdot\cos(nt), \quad t \in [0, 2\pi], n \in \mathbf{N}$

4) Să se reprezinte „Clopotul lui GAUSS" exprimat de funcţia de două variabile:

$$f(x,y) = \frac{1}{\sqrt{2\pi}}\,e^{-\frac{1}{2}(x^2+y^2)}.$$

7.6. Bibliografie

[1] M. VLADA, A. POSEA
Grafica automată în limbajul FORTRAN 77 şi aplicaţii, Tipografia Universităţii din Bucureşti, 1990.

[2] L. NICOLESCU
Curs de geometrie, Tipografia Universităţii din Bucureşti, 1989.

[3] D. SMARANDA
Elemente de teoria curbelor şi suprafeţelor, Tipografia Universităţii din Bucureşti, 1984.

[4] C. UDRIŞTE
Curbe şi suprafeţe, Tipografia Institutului Politehnic Bucureşti, 1974.

[5] C. UDRIŞTE, C. RADU, C. DICU, O. MALANCIOIU
Probleme de algebră, geometrie şi ecuaţii diferenţiale, Editura Didactică şi Pedagogică, Bucureşti, 1981.

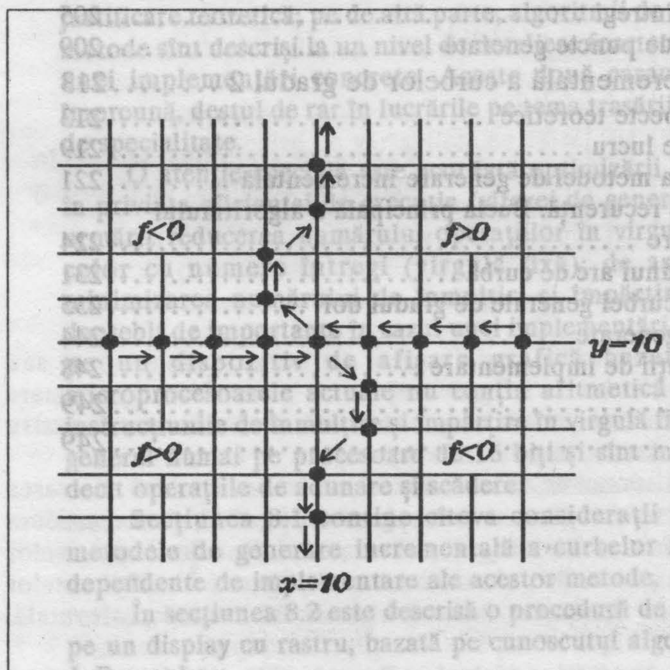
[6] S. SBURLAN
Principiile fundamentale ale matematicii moderne. Lecţii de analiză matematică, Editura Academiei Române, 1991.

[7] T. POSTON, I. STEWART
Teoria catastrofelor şi aplicaţii, Editura Tehnică, Bucureşti, 1985.

[8] D. DOGARU
Metode noi în proiectare. Elemente de grafică 3-D, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1988.

[9] D. THALMANN
Informatique graphique: concepts et techniques avec le langage MIRA, éditeur Gaëtan Morin, 1983.

TRASAREA INCREMENTALĂ A CURBELOR DE GRADUL ÎNTÎI ȘI DOI



8

8

În cele ce urmează vom prezenta cîteva metode incrementale de generare a curbelor plane pentru dispozitive de afişare grafică cu tub catodic, cu rastru dreptunghiular. Metodele incrementale se dovedesc deosebit de eficiente pentru curbele de gradul 1 (linii drepte) şi 2 (cercuri, elipse, hiperbole, parabole). În general, viteza de generare a dreptelor (vectorilor) este determinantă pentru performanţele globale ale unui dispozitiv periferic de afişare grafică, mai ales în condiţii de utilizare interactivă. De asemenea, pentru aplicaţiile mai evoluate din domeniul proiectării asistate de calculator, posibilitatea trasării eficiente a curbelor de gradul doi este de cea mai mare importanţă.

Metodele incrementale prezentate sînt însoţite de o riguroasă justificare teoretică; pe de altă parte, algoritmii de trasare bazaţi pe aceste metode sînt descrişi la un nivel de detaliere foarte apropiat de necesităţile unei implementări concrete. Aceste două caracteristici sînt întîlnite, împreună, destul de rar în lucrările pe tema trasării curbelor din literatura de specialitate.

O atenţie specială este acordată optimizării algoritmilor prezentaţi în privinţa eficienţei de execuţie (vitezei de generare). În acest scop, s-a urmărit reducerea numărului operaţiilor în virgulă mobilă, în favoarea celor cu numere întregi (virgulă fixă); de asemenea, s-a urmărit minimizarea numărului de înmulţiri şi împărţiri. Aceste aspecte sînt deosebit de importante în cazul unei implementări concrete a algoritmilor pe un dispozitiv de afişare grafică bazat pe microprocesor: microprocesoarele actuale nu conţin aritmetică în virgulă mobilă, iar instrucţiunile de înmulţire şi împărţire în virgulă fixă sînt implementate în general numai pe procesoare de 16 biţi şi sînt mult mai puţin eficiente decît operaţiile de adunare şi scădere.

Secţiunea 8.1 conţine cîteva consideraţii generale cu privire la metodele de generare incrementală a curbelor şi precizează aspectele dependente de implementare ale acestor metode.

În secţiunea 8.2 este descrisă o procedură de trasare a liniilor drepte pe un display cu rastru, bazată pe cunoscutul algoritm de generare al lui J. Bresenham.

În secţiunea 8.3 este analizat şi prezentat un algoritm de trasare a

cercurilor definite prin coordonatele centrului şi rază, în ipoteza că aceste numere sînt multipli întregi de unităţi de rastru. De asemenea, în secţiunea 8.3.5 este descris un algoritm de trasare asemănător pentru cazul în care coordonatele centrului şi raza sînt numere semiîntregi.

În secţiunea 8.4 este abordată problema generală a trasării unei curbe plane de gradul doi, pornind de la reprezentarea analitică obişnuită a unei astfel de curbe: o ecuaţie de gradul doi în coordonatele carteziene x şi y. Este indicată o metodă de mare eficienţă care utilizează în bucla principală de trasare numai operaţii de adunare, scădere şi comparaţie între numere întregi. Această metodă reprezintă o generalizare naturală a procedurilor de trasare descrise în secţiunile 8.2 şi 8.3.

8.1. Generalităţi

În ultimul timp, în domeniul graficii interactive cu ajutorul calculatorului s-a generalizat utilizarea dispozitivelor de afişare cu tub catodic, cu rastru dreptunghiular. Aceste dispozitive se caracterizează printr-un spaţiu de afişare alcătuit dintr-o matrice rectangulară de elemente grafice indivizibile, numite puncte sau pixeli (fig. 8-1).

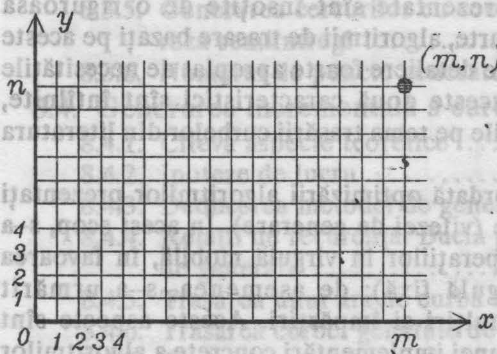


Figura 8-1. Punctul de coordonate (m, n) în spaţiul de afişare al unui display cu rastru.

Pentru reprezentarea unei imagini pe ecran, calculatorul sau procesorul dispozitivului de afişare trebuie să genereze o aproximare discretizată cît mai fidelă pentru imaginea ideală, şi să selecteze în rastru punctele corespunzătoare imaginii discretizate.

Vom adopta sistemul de coordonate cel mai natural pentru adresarea punctelor de pe ecran; după cum este sugerat în fig. 8-1, vom considera originea $(0, 0)$ în punctul din stînga jos al ecranului, cu valorile absciselor crescînd la dreapta şi valorile ordonatelor crescînd în sus. Unităţile axelor Ox, Oy sînt egale cu distanţele între doi pixeli alăturaţi pe orizontală, respectiv verticală.

Pentru multe dispozitive reale de afişare grafică, pasul rastrului pe orizontală diferă de pasul pe verticală. În aceast capitol vom face

presupunerea că unitățile rastrului sînt egale pe orizontală și verticală, ceea ce simplifică algoritmii de trasare prezentați. Această presupunere nu este esențială pentru algoritmii de trasare a dreptelor (secțiunea 8.2) și a unei curbe generale de gradul doi (secțiunea 8.4), deoarece o dilatare pe una din axe revine la o modificare a parametrilor care definesc aceste curbe, fără schimbarea naturii curbei trasate. În cazul algoritmului de generare a cercurilor (secțiunea 8.3), ipoteza de echidistanță a rastrului este esențială.

De asemenea, deoarece în acest capitol ne interesează numai forma curbelor trasate, vom presupune că fiecare pixel de pe ecran are două stări posibile: aprins sau stins. Vom neglija celelalte atribute posibile pentru reprezentarea unei imagini pe ecran, cum sînt culoarea, nuanța, intensitatea, etc.

În cele ce urmează, pentru generarea efectivă a unui punct pe ecran vom folosi operații de forma:

call plot (x,y)

unde *plot* este o subrutină neprecizată, care depinde de configurația hardware a dispozitivului de afișare, iar x,y sînt coordonatele întregi în spațiul de afișare grafică pentru punctul care trebuie generat. Pentru coordonatele x,y vom folosi denumirea de *variabile de adresare fizică*. În general, algoritmii de trasare incrementală prezentați în continuare efectuează deplasări elementare între două puncte vecine pe orizontală, verticală sau diagonală, care se traduc prin operații simple de incrementare sau decrementare a variabilelor de adresare fizică.

Spre deosebire de variabilele de stare, variabilele de adresare fizică utilizate în continuare au mai mult rol ilustrativ, pentru descrierea funcționării algoritmului. În cazul unei implementări concrete a unui algoritm de trasare, programatorul trebuie să aleagă cele mai potrivite posibilități de adresare grafică în funcție de configurația hardware pe care se face implementarea. De exemplu, pentru multe display-uri grafice punctele rastrului sînt reprezentate printr-o zonă continuă de memorie, fiecare octet păstrînd starea curentă pentru 8 pixeli adiacenți pe orizontală. În acest caz, se pot alege ca variabile de adresare a unui punct pe ecran adresa în memorie a octetului care conține punctul respectiv, și o mască binară de 8 biți pentru selectarea bitului corespunzător.

Performanțele finale ale unui algoritm de trasare depind în mare masură de eficiența rutinei *plot* deoarece, cum se va vedea în continuare, celelalte operații executate sînt în general foarte simple: adunări și scăderi de numere întregi și testări de semn pentru variabile întregi. De aceea, rutina *plot* trebuie proiectată astfel încît, pe de o parte, operația fizică de generare a unui punct, și pe de altă parte, deplasarea în rastru la un punct alăturat, să se efectueze cît mai rapid posibil.

Tehnicile de trasare incrementală a curbelor reprezintă o formă de calcul a punctelor generate succesiv, în care fiecare pas iterativ este simplificat pe baza unor informaţii referitoare la punctul generat anterior. Algoritmul de trasare păstrează în permanenţă informaţii cu privire la starea curentă a procesului iterativ, prin intermediul uneia sau mai multor *variabile de stare*. Acestea trebuie alese astfel încît, pe de o parte, să permită realizarea cît mai simplă a funcţiilor algoritmului, şi pe de altă parte să poată fi actualizate cît mai uşor în vederea trecerii la următorul pas iterativ. De multe ori, variabilele de stare sînt redundante, pentru a da posibilitatea implementării cu maximum de eficienţă a operaţiilor necesare la fiecare iteraţie.

Pentru trasarea curbelor plane de gradul întîi şi doi pe dispozitive de afişare cu rastru rectangular, metodele incrementale sînt foarte potrivite din două motive:

- din cauza structurii discrete a mediului de afişare;
- există posibilitatea implementării cu mijloace relativ simple. Într–adevăr, metodele de trasare incrementală se bazează pe scheme cu diferenţe între mărimi asociate punctelor succesive, care în cazul curbelor de grad cel mult doi se traduc prin relaţii liniare de transformare iterativă a variabilelor de stare. Aceste relaţii liniare pot fi implementate în general prin operaţii simple de adunare şi scădere de numere întregi; prin urmare, metodele incrementale sînt accesibile celor mai simple microprocesoare actuale.

Algoritmii de trasare prezentaţi în aceast capitol, pornind de la nişte parametri care precizează analitic o curbă de gradul întîi sau doi, construiesc în general cea mai bună reprezentare discretizată a curbei respective. Ţinînd seama de faptul că modelul reprezentării este o curbă continuă şi pentru care tangenta variază continuu, dar spaţiul de afişare este discret, în dezvoltarea algoritmilor vom adopta următoarele convenţii metodologice de reprezentare:

- Pentru fiecare punct generat la un capăt al curbei, sau la o margine a ferestrei de vizualizare, există exact un punct vecin pe orizontală, verticală sau în diagonală, generat pe curbă.
- Oricare alt punct generat pe curbă are exact două puncte vecine pe orizontală, verticală sau în diagonală, generate pe curbă, cu care formează un unghi de 180° sau 135°.

Aceste convenţii vor fi referite în continuare sub numele de *regula de conexiune discretă*. Toţi algoritmii prezentaţi generează succesiuni de puncte care îndeplinesc această regulă.

De exemplu, regula de conexiune discretă este îndeplinită de curba din fig. 8–2*a*, dar nu este îndeplinită de curbele din fig. 8–2*b*, 8–2*c* şi 8–2*d*.

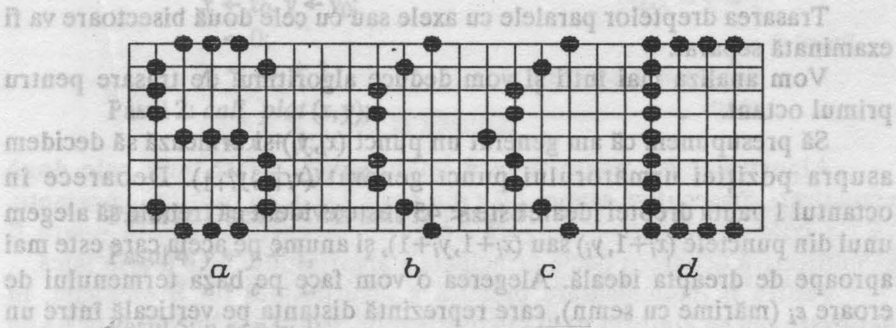


Figura 8–2.
Numai curba *a* îndeplinește condiția de conexiune discretă.

8.2. Generarea incrementală a liniilor drepte

Problema concretă a trasării liniilor drepte pe un display cu rastru se pune în modul următor: dîndu–se două puncte (x_0, y_0) și (x_f, y_f) în spațiul de afișare, se cere să se selecteze acei pixeli din rastru care aproximează cel mai bine segmentul de linie dreaptă ideală dintre cele două puncte (fig. 8–3). Șirul de puncte generate trebuie să îndeplinească condiția de conexiune discretă precizată în secțiunea 8.1.

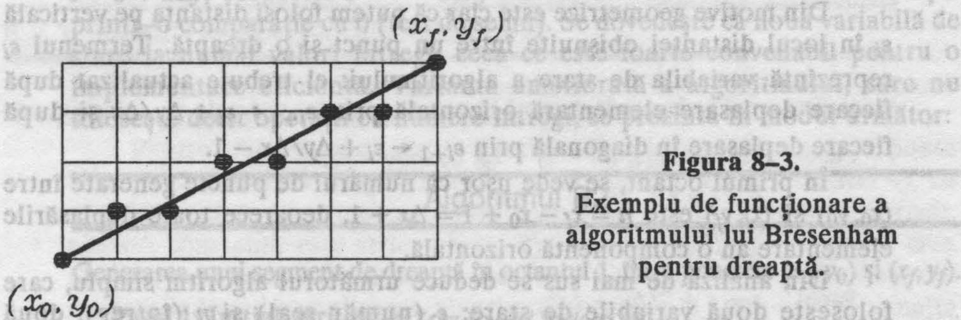


Figura 8–3.
Exemplu de funcționare a algoritmului lui Bresenham pentru dreaptă.

Se poate obține un algoritm eficient de trasare urmînd metoda descrisă de J. Bresenham [1]. Se notează $\Delta x = x_f - x_0$, $\Delta y = y_f - y_0$ și se poate presupune $\Delta y \geq 0$, inversînd la nevoie punctele inițial și final (x_0, y_0) și (x_f, y_f). În funcție de înclinarea dreptei care trebuie trasată, se deosebesc 4 cazuri:

- $0 < \Delta y < \Delta x$ (trasare în octantul 1)
- $0 < \Delta x < \Delta y$ (octantul 2)
- $0 < -\Delta x < \Delta y$ (octantul 3)
- $0 < \Delta y < -\Delta x$ (octantul 4)

Trasarea dreptelor paralele cu axele sau cu cele două bisectoare va fi examinată separat.

Vom analiza mai întîi și vom deduce algoritmul de trasare pentru primul octant.

Să presupunem că am generat un punct (x_i, y_i) și urmează să decidem asupra poziției următorului punct generat (x_{i+1}, y_{i+1}). Deoarece în octantul 1 panta dreptei ideale este < 45°, este evident că trebuie să alegem unul din punctele (x_i+1, y_i) sau (x_i+1, y_i+1), și anume pe acela care este mai aproape de dreapta ideală. Alegerea o vom face pe baza termenului de eroare ε_i (mărime cu semn), care reprezintă distanța pe verticală între un punct generat (x_i, y_i) și dreapta ideală (fig. 8–4)

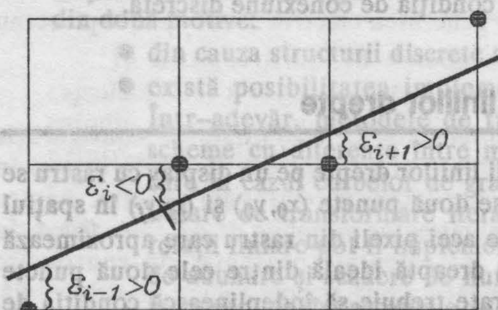


Figura 8–4.

Termeni de eroare pozitivi și negativi pentru algoritmul lui Bresenham.

Din motive geometrice este clar că putem folosi distanța pe verticală ε_i în locul distanței obișnuite între un punct și o dreaptă. Termenul ε_i reprezintă variabila de stare a algoritmului: el trebuie actualizat după fiecare deplasare elementară orizontală prin $\varepsilon_{i+1} \leftarrow \varepsilon_i + \Delta y/\Delta x$ și după fiecare deplasare în diagonală prin $\varepsilon_{i+1} \leftarrow \varepsilon_i + \Delta y/\Delta x - 1$.

În primul octant, se vede ușor că numărul de puncte generate între (x_0, y_0) și (x_f, y_f) este $n = x_f - x_0 + 1 = \Delta x + 1$, deoarece toate deplasările elementare au o componentă orizontală.

Din analiza de mai sus se deduce următorul algoritm simplu, care folosește două variabile de stare: ε (număr real) și n (întreg), două variabile de adresare fizică x și y, precum și două constante întregi Δx și Δy (constante în sensul că depind numai de punctele inițiale, dar nu se modifică în cursul execuției).

Algoritmul B1

Generarea unui segment de dreaptă în octantul 1, între punctele (x_0, y_0) și (x_f, y_f).

Pasul 1: Inițializări: $\Delta x \leftarrow x_f - x_0$, $\Delta y \leftarrow y_f - y_0$;

$x \leftarrow x_0, y \leftarrow y_0;$
$\varepsilon \leftarrow 0;$
$n \leftarrow \Delta x + 1.$
Pasul 2: *call plot* $(x, y);$
$x \leftarrow x + 1;$
$\varepsilon \leftarrow \varepsilon + \Delta y / \Delta x,$
Pasul 3: Dacă $\varepsilon < 1/2$, salt la Pasul 5.
Pasul 4: $y \leftarrow y + 1;$
$\varepsilon \leftarrow \varepsilon - 1.$
Pasul 5: $n \leftarrow n - 1;$
dacă $n \neq 0$, salt la Pasul 2;
stop.

Algoritmul B1 prezintă două dezavantaje: utilizează o variabilă reală și necesită o operație de împărțire. Acestea pot fi evitate dacă în locul variabilei de stare ε introducem:

$$s = (\varepsilon - 1/2)\, 2\Delta x\,.$$

Funcționarea algoritmului nu este afectată de această schimbare liniară de variabilă, cu condiția să modificăm corespunzător toate operațiile în care este implicată variabila de stare. Astfel, inițializarea $\varepsilon \leftarrow 0$ se înlocuiește cu $s \leftarrow -\Delta x$, actualizările lui ε din pașii 2 și 4 se înlocuiesc prin $s \leftarrow s + 2\Delta y$, respectiv $s \leftarrow s - 2\Delta x$, iar testul din Pasul 3 se înlocuiește printr–o comparație cu 0 (test de semn). Se dovedește că noua variabilă de stare ia numai valori întregi, ceea ce este foarte convenabil pentru o implementare eficientă. Varianta ameliorată a algoritmului, care nu folosește decît operații cu numere întregi, se prezintă în modul următor:

Algoritmul B2

Generarea unui segment de dreaptă în octantul 1, între punctele (x_0, y_0) și (x_f, y_f).

Pasul 1: Inițializări: $\Delta x \leftarrow x_f - x_0$, $\Delta y \leftarrow y_f - y_0;$
$x \leftarrow x_0, y \leftarrow y_0;$
$s \leftarrow -\Delta x;$
$n \leftarrow \Delta x + 1.$
Pasul 2: *call plot* $(x, y);$
$x \leftarrow x + 1;$
$s \leftarrow s + 2\Delta y.$
Pasul 3: Dacă $s < 0$, salt la Pasul 5.
Pasul 4: $y \leftarrow y + 1;$

$s \leftarrow s - 2\Delta x$.

Pasul 5: $n \leftarrow n - 1$;
dacă $n \neq 0$, salt la Pasul 2;
stop.

Algoritmul B2 poate fi încă îmbunătăţit, astfel încît cele două actualizări consecutive asupra lui *s* din paşii 2 şi 4, cu cantităţi constante, să fie combinate într-o singură operaţie. Această reducere poate fi semnificativă deoarece bucla principală a algoritmului este foarte mică.

Pentru o implementare practică a algoritmului lui Bresenham, în hardware sau cu ajutorul unui microprocesor, recomandăm varianta B3 de mai jos, care la o examinare atentă se dovedeşte echivalentă cu varianta B2. Algoritmul B3 este mai puţin compact decît B2, dar în acelaşi timp este mai eficient, din următoarele motive:

- actualizarea lui *s* se face o singură dată la o trecere prin bucla principală, prin adăugarea unei cantităţi constante;
- testarea semnului lui *s* are loc imediat după o operaţie aritmetică asupra lui *s*, ceea ce este foarte convenabil din punct de vedere al programării pe un microprocesor;
- salturile necondiţionate în interiorul buclei principale sînt evitate prin dublarea operaţiilor comune între cele două ramuri ale buclei principale.

Algoritmul B3

Generarea unui segment de dreaptă în octantul 1, între punctele (x_0, y_0) şi (x_f, y_f).

Pasul 1: Iniţializări: $\Delta x \leftarrow x_f - x_0$, $\Delta y \leftarrow y_f - y_0$;
$x \leftarrow x_0$, $y \leftarrow y_0$;
$s \leftarrow \Delta x$;
$n \leftarrow \Delta x + 1$.

Pasul 2: $s \leftarrow s + 2(\Delta y - \Delta x)$;
dacă $s \geq 0$, salt la Pasul 6.

Pasul 3: *call plot* (x, y);
$x \leftarrow x + 1$.

Pasul 4: $n \leftarrow n - 1$;
dacă $n = 0$, stop.

Pasul 5: $s \leftarrow s + 2\Delta y$;
dacă $s < 0$, salt la Pasul 3.

Pasul 6: *call plot* (x, y);

$x \leftarrow x + 1$;
$y \leftarrow y + 1$.

Pasul 7: $n \leftarrow n - 1$;
dacă $n \neq 0$, salt la Pasul 2;
stop.

În algoritmul de mai sus, este de la sine înțeles că valorile constante $2(\Delta y - \Delta x)$ și $2\Delta y$, cu care se actualizează s, se vor calcula o singură dată în faza de inițializare. Aceste operații nu au mai fost explicitate, pentru a ușura urmărirea funcționării algoritmului.

După dezvoltarea algoritmului de trasare pentru octantul 1, este ușor de văzut ce modificări sînt necesare pentru celelalte cazuri (octanții 2, 3 și 4). Pentru octantul 2, algoritmul de trasare se obține din B3, interschimbînd rolurile celor două axe de coordonate. Pentru octantul 4, inițializarea constantei (pozitive) Δx trebuie efectuată în forma $\Delta x \leftarrow x_0 - x_f$, în plus, cele două incrementări ale lui x din pașii 3 și 6, corespunzînd deplasărilor elementare pe orizontală sau în diagonală, trebuie înlocuite prin decrementări $x \leftarrow x - 1$ (în octantul 4, deplasările elementare sînt la stînga sau stînga–sus). Algoritmul pentru octantul 3 se poate obține fie din cel pentru octantul 4, inversînd rolurile axelor x și y, fie din algoritmul pentru octantul 2, schimbînd semnul lui Δx la inițializare și direcția deplasărilor diagonale din dreapta–sus în stînga–sus.

8.2.1. Cazuri speciale, optimizări

Să examinăm acum cazurile speciale cînd dreapta care trebuie trasată este orizontală, verticală sau înclinată la ±45°.

Pentru trasarea unei drepte paralele cu una din bisectoare, se poate folosi algoritmul B3 (respectiv varianta modificată pentru octantul 4), cu observația că în acest caz constanta $2(\Delta y - \Delta x)$ devine 0 (în mod normal, această constantă este strict negativă). Această împrejurare poate crea efecte opuse celui dorit la testul din Pasul 2 al algoritmului. În acest caz nu este indicată complicarea algoritmului pentru a trata corect situația cînd $\Delta x = \Delta y$, deoarece se reduce eficiența execuției în cazul general; se recomandă adăugarea unei secvențe speciale pentru generarea dreptelor înclinate la ±45°. Această secvență este foarte simplă: se generează $\Delta x + 1$ puncte separate prin deplasări elementare în diagonală (nu este necesară variabila de stare s).

Dreptele orizontale sau verticale sînt trasate corect de algoritmul B3, respectiv de varianta modificată pentru octantul 2. Aceste cazuri pot fi implementate prin secvențe speciale, în eventualitatea că se poate beneficia de organizarea memoriei video a display–ului, pentru mărirea vitezei de generare. De exemplu, pentru multe display–uri, un octet în

memoria video reprezintă 8 pixeli adiacenți pe orizontală. Algoritmul B3 face 8 accese la același octet din memoria video pentru a genera 8 puncte. Secvența specială pentru trasarea dreptelor orizontale poate genera 8 puncte simultan printr–un singur acces la memoria video.

Pentru acest tip de display–uri, aceeași idee de minimizare a numărului acceselor la memoria video se poate aplica pentru generarea dreptelor înclinate în octantul 1 (algoritmul B3) sau în octantul 4. Deoarece pantele dreptelor sînt mai mici de 45°, în cursul generării apar grupuri de 2 sau mai multe puncte consecutive adiacente pe orizontală. Punctele vecine reprezentate binar în același octet de memorie video, pot fi generate printr–un singur acces la acel octet de memorie. De exemplu, pentru trasarea unei linii drepte între punctele (0, 0) și (23, 4) sînt necesare numai 7 accese la memorie, prin care sînt generate 24 puncte (fig. 8–5).

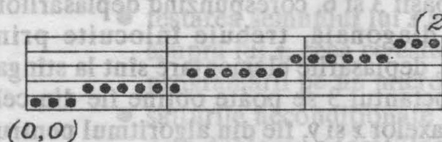


Figura 8–5.

În octantul 1, numărul acceselor necesare la memoria video este mai mic decît numărul punctelor generate.

Această idee de optimizare reclamă complicarea algoritmului B3. Este necesară o variabilă suplimentară *c* pentru acumularea punctelor adiacente pe orizontală; de asemenea, este necesară modificarea subrutinei *plot*.

Evident, optimizarea descrisă mai sus nu este efectivă decît dacă accesul la variabila suplimentară *c* este sensibil mai rapid decît accesul la memoria video. Este foarte indicată plasarea variabilei *c* într–unul din registrele generale, alături de celelalte variabile ale algoritmului; această posibilitate depinde de (micro)procesorul pe care se implementează algoritmul.

8.3. Generarea incrementală a cercurilor

În această secțiune vom prezenta o metodă incrementală de generare a cercurilor cu centrul într–un punct din rastru și cu raza un număr întreg de unități de rastru. De asemenea, în secțiunea 8.3.5 vom descrie o metodă incrementală de generare în cazul cînd coordonatele centrului și raza sînt numere semiîntregi.

Sînt considerate criterii de minimizare liniară și pătratică a erorii, care să asigure cea mai bună aproximare a conturului ideal printr–o succesiune de puncte de coordonate întregi. În algoritmul rezultat, bucla

principală de generare incrementală a punctelor succesive utilizează numai operații simple de adunare, scădere și testare a semnului pentru numere întregi; întregul algoritm poate fi implementat numai cu aritmetică în virgulă fixă, și nu necesită operații de înmulțire și împărțire, ceea ce este oarecum surprinzător deoarece curba generată este de gradul doi. Simplitatea operațiilor executate de algoritm dă posibilitatea implementării eficiente pe cele mai simple microprocesoare actuale.

În fig. 8–6 este prezentat un exemplu de funcționare a algoritmului de trasare în primul cadran al axelor de coordonate.

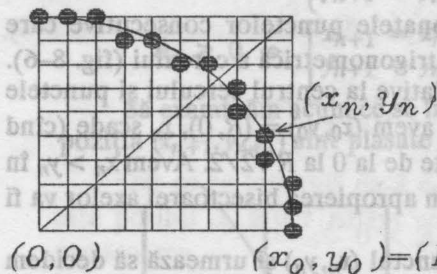


Figura 8–6.

Generarea prin puncte a unui sfert de cerc de rază $R = 9$.

Algoritmul de generare a cercurilor se bazează în mod esențial pe ipoteza de echidistanță orizontală și verticală a punctelor rastrului, discutată în secțiunea 8.1. Această presupunere nu este necesară pentru algoritmul de trasare a liniilor drepte, deoarece o dreaptă rămîne tot dreaptă în urma unei dilatări pe axa *Ox* sau *Oy*, dar, evident, această invarianță nu este valabilă pentru cercuri. Dacă condiția de echidistanță a rastrului nu este îndeplinită, algoritmul descris în continuare va trasa elipse în loc de cercuri.

Este posibilă abordarea cazului general, în care dimensiunile celulei elementare din rastru sînt în raportul $p/q \neq 1$ (p, q întregi), cum se întîlnește la multe dispozitive reale. În acest caz, analiza algoritmului se complică și aspectele esențiale devin mai obscure, iar în final eficiența în execuție (viteza de generare) scade. De aceea vom prefera să prezentăm algoritmul de generare a cercurilor în cazul simplu $p/q = 1$; pe de altă parte, generarea elipselor și a altor curbe de gradul doi este discutată în secțiunea 8.4.

La fel ca în cazul algoritmului de generare a dreptelor, vom examina problema separat pentru fiecare octant. Din geometria cercului rezultă că, în interiorul unui octant, sînt posibile numai două deplasări elementare între două puncte generate consecutiv: o deplasare perpendiculară pe axa mai apropiată, sau o deplasare diagonală. De exemplu (fig. 8–6), în octantul 1 deplasările elementare sînt în sus sau spre stînga–sus, iar în octantul 2 la stînga sau spre stînga–sus. Prin urmare, după generarea unui punct, există numai două posibilități de alegere a următorului punct generat; aceasta creează perspectiva dezvoltării unui algoritm eficient cu mijloace relativ reduse.

8.3.1. Analiza algoritmului de generare în primul octant

Algoritmul de trasare incrementală a cercului trebuie să calculeze coordonatele punctelor consecutive care urmează a fi generate, prin parcurgerea cercului într–un anumit sens, de exemplu sensul trigonometric direct. Pentru aflarea coordonatelor unui punct se folosesc coordonatele precedentului punct generat şi anumite formule de recurenţă liniare. Pentru deducerea acestor formule de recurenţă, să analizăm trasarea incrementală a unui arc de cerc de rază R cu centrul în origine, situat în primul octant al axelor de coordonate.

Notăm cu (x_n, y_n), $n \geq 0$ coordonatele punctelor consecutive care urmează a fi generate, în parcurgerea trigonometrică a cercului (fig. 8–6). Deoarece coordonatele (x_n, y_n) sînt relative la centrul cercului şi punctele respective se găsesc în primul octant, avem $(x_0, y_0) = (R, 0)$, x_n scade (cînd n creşte) de la R la $R\sqrt{2}/2$, iar y_n creşte de la 0 la $R\sqrt{2}/2$. Avem $x_n > y_n$ în interiorul octantului 1; comportarea în apropierea bisectoarei axelor va fi examinată separat în continuare.

Să presupunem că am generat punctul (x_n, y_n) şi urmează să decidem asupra poziţiei următorului punct care trebuie generat (x_{n+1}, y_{n+1}). Deoarece ne găsim în primul octant, este clar că următorul punct generat are ordonata $y_{n+1} = y_n+1$, dar abscisa sa x_{n+1} poate fi fie x_n, fie x_n-1 (tangenta la cercul ideal face cu verticala un unghi între 0 şi 45°, la stînga acesteia).

Pentru a alege între cele două posibilităţi, $(x_{n+1}, y_{n+1}) = (x_n, y_n+1)$ sau $(x_{n+1}, y_{n+1}) = (x_n-1, y_n+1)$, vom considera următorul criteriu, evident de altfel: va fi generat punctul care este situat cel mai aproape de cercul ideal (nu pot fi generate ambele puncte din cauza condiţiei de conexiune discretă discutată în secţiunea 8.1). Mai precis, va fi generat punctul pentru care distanţa pînă la cerc $|r_{n+1} - R|$ este minimă (am notat $r_n = \sqrt{x_n^2 + y_n^2}$, $n \geq 0$).

Introducem notaţiile:

$$D_{1n} = \sqrt{x_n^2 + (y_n+1)^2} - R \tag{3.1}$$

$$D_{2n} = R - \sqrt{(x_n-1)^2 + (y_n+1)^2}\,, \quad n \geq 0\,.$$

Prin urmare, dacă $|D_{1n}| < |D_{2n}|$, vom alege punctul $(x_{n+1}, y_{n+1}) = (x_n, y_n+1)$ pentru a fi generat, iar dacă $|D_{1n}| \geq |D_{2n}|$, vom alege $(x_{n+1}, y_{n+1}) = (x_n-1, y_n+1)$.

Să presupunem acum că cele 2 puncte candidat sînt plasate de o parte şi de alta a cercului ideal, adică (x_n, y_n+1) este în exteriorul cercului, iar

(x_n-1, y_n+1) în interiorul acestuia. În acest caz, este evident că $D_{1n} \geq 0$, $D_{2n} \geq 0$ și putem decide asupra punctului care va fi generat în urma comparației cu 0 a diferenței $D_{1n} - D_{2n}$, cantitate pe care o notăm cu δ_n:

$$\delta_n = D_{1n} - D_{2n} = \sqrt{x_n^2 + (y_n+1)^2} + \sqrt{(x_n-1)^2 + (y_n+1)^2} - 2R. \quad (3.2)$$

Prin urmare, dacă $\delta_n < 0$ vom genera punctul (x_n, y_n+1), iar dacă $\delta_n \geq 0$ vom genera punctul (x_n-1, y_n+1):

$$\begin{aligned} \delta_n < 0 &\Rightarrow \begin{cases} x_{n+1} = x_n \\ y_{n+1} = y_n+1 \end{cases} \\ \delta_n \geq 0 &\Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n+1. \end{cases} \end{aligned} \quad (3.3)$$

Să examinăm acum ce se întîmplă dacă cele 2 puncte candidat pentru poziția (x_{n+1}, y_{n+1}) sînt plasate de aceeași parte a cercului ideal.

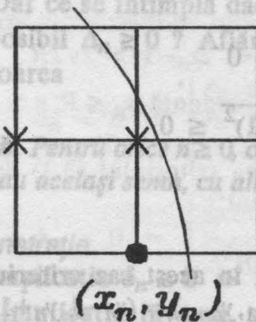


Figura 8–7.
Ambele puncte candidat pentru poziția (x_{n+1}, y_{n+1}) sînt în interiorul cercului.

Dacă ambele puncte sînt plasate în interiorul cercului (fig. 8–7), atunci:

$$R \geq \sqrt{x_n^2 + (y_n+1)^2} > \sqrt{(x_n-1)^2 + (y_n+1)^2}. \quad (3.4)$$

Prin urmare, în acest caz:

$$|D_{1n}| = R - \sqrt{x_n^2 + (y_n+1)^2}$$

$$|D_{2n}| = R - \sqrt{(x_n-1)^2 + (y_n+1)^2}$$

și din (3.4) rezultă $|D_{1n}| < |D_{2n}|$. Vom alege deci punctul $(x_{n+1}, y_{n+1}) = (x_n, y_n+1)$ pentru a fi generat; de altfel, din fig. 8–7 este clar că punctul (x_n, y_n+1) este mai aproape de cercul ideal decît (x_n-1, y_n+1).

Pe de altă parte, din (3.1) și (3.4) deducem $D_{1n} \leq 0$, $D_{2n} > 0$ și deci $\delta_n = D_{1n} - D_{2n} < 0$. Prin urmare, criteriul (3.3) ne conduce la decizia corectă în acest caz.

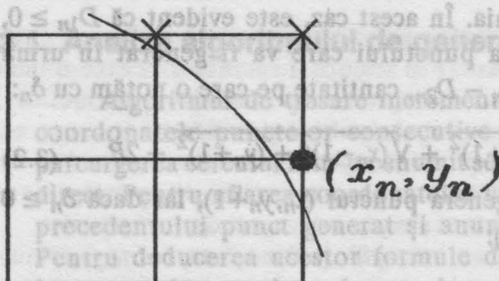


Figura 8–8. Ambele puncte candidat pentru poziţia (x_{n+1}, y_{n+1}) sînt în exteriorul cercului.

Dacă ambele puncte candidat sînt plasate în afara cercului (fig. 8–8), atunci:

$$R \le \sqrt{(x_n-1)^2 + (y_n+1)^2} < \sqrt{x_n^2 + (y_n+1)^2}.$$

Deducem succesiv:

$$D_{1n} = \sqrt{x_n^2 + (y_n+1)^2} - R > 0$$

$$D_{2n} = R - \sqrt{(x_n-1)^2 + (y_n+1)^2} \le 0$$

$$|D_{1n}| > |D_{2n}|$$

$$\delta_n = D_{1n} - D_{2n} > 0.$$

Ultimele două inegalităţi dovedesc că şi în acest caz criteriul (3.3) ne conduce la decizia corectă de a alege $(x_{n+1}, y_{n+1}) = (x_n-1, y_n+1)$.

Prin urmare, am arătat pînă acum că putem folosi întotdeauna criteriul (3.3) pentru alegerea corectă a punctului care urmează să fie generat (x_{n+1}, y_{n+1}), dacă pornim de la punctul generat anterior (x_n, y_n), presupus corect plasat.

Cantităţile $\delta_n, n \ge 0$ conţin radicali, prin urmare sînt neconvenabile dacă se urmăreşte o implementare eficientă, mai ales pe o maşină care nu dispune decît de aritmetică în virgulă fixă. Ar fi foarte avantajos să se înlocuiască, dacă este posibil, criteriul (3.3) de decizie cu un criteriu echivalent, dar care să necesite numai operaţii cu numere întregi.

În locul numărului $\delta_n = \sqrt{x_n^2 + (y_n+1)^2} + \sqrt{(x_n-1)^2 + (y_n+1)^2} - 2R$, să examinăm cantitatea:

$$\begin{aligned} \Delta_n &= x_n^2 + (y_n+1)^2 + (x_n-1)^2 + (y_n+1)^2 - 2R^2 = \\ &= x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 \end{aligned} \tag{3.5}$$

care se obţine din δ_n înlocuind fiecare termen al sumei prin pătratul său.

Se arată uşor că $\delta_n \geq 0 \Rightarrow \Delta_n \geq 0$. Într-adevăr,

$$\delta_n \geq 0 \Rightarrow \sqrt{x_n^2 + (y_n+1)^2} + \sqrt{(x_n-1)^2 + (y_n+1)^2} \geq 2R \Rightarrow$$

$$\Rightarrow x_n^2 + (y_n+1)^2 + (x_n-1)^2 + (y_n+1)^2 +$$
$$+ 2\sqrt{[x_n^2 + (y_n+1)^2][(x_n-1)^2 + (y_n+1)^2]} \geq 4R^2 \Rightarrow$$
$$\Rightarrow 2[x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2] \geq$$
$$\geq x_n^2 + (y_n+1)^2 + (x_n-1)^2 + (y_n+1)^2 -$$
$$- 2\sqrt{[x_n^2 + (y_n+1)^2][(x_n-1)^2 + (y_n+1)^2]} \geq 0 \Rightarrow$$
$$\Rightarrow 2\Delta_n \geq 0\,.$$

Dar ce se întîmplă dacă $\delta_n < 0$? Rezultă $\Delta_n < 0$ şi în acest caz, sau este posibil $\Delta_n \geq 0$? Aflăm un răspuns fericit la această întrebare din următoarea

Teoremă. *Pentru orice $n \geq 0$, cantităţile δ_n şi Δ_n definite în (3.2) respectiv (3.5) au aceleaşi semn, cu alte cuvinte $\delta_n \geq 0 \Leftrightarrow \Delta_n \geq 0$.*

Demonstraţie.

Implicaţia $\delta_n \geq 0 \Rightarrow \Delta_n \geq 0$ a fost deja dovedită. Vom demonstra acum implicaţia inversă, sau, ceea ce este echivalent, vom dovedi că inegalităţile $\delta_n < 0, \Delta_n \geq 0$ nu sînt posibile simultan.

Să presupunem deci $\delta_n < 0, \Delta_n \geq 0$, adică:

$$x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 + 2\sqrt{[x_n^2 + y_{n+1}^2][(x_n-1)^2 + y_{n+1}^2]} < 4R^2$$
$$x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 - 2R^2 \geq 0$$

(am scris y_{n+1} în loc de y_n+1). Rescriem aceste inegalităţi astfel:

$$2\sqrt{x_n^2 (x_n-1)^2 + x_n^2 y_{n+1}^2 + (x_n-1)^2 y_{n+1}^2 + y_{n+1}^4} <$$
$$< 4R^2 - x_n^2 - (x_n-1)^2 - 2y_{n+1}^2 \tag{3.6}$$
$$x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 - 2R^2 > 0$$

(în a doua inegalitate, semnul egal nu este posibil deoarece în membrul stîng este un număr întreg impar). Să ridicăm la pătrat prima inegalitate:

$$4[x_n^2 (x_n-1)^2 + x_n^2 y_{n+1}^2 + (x_n-1)^2 y_{n+1}^2 + y_{n+1}^4] <$$

$$< 16R^4 + x_n^4 + (x_n-1)^4 + 4y_{n+1}^4 - 8R^2x_n^2 - 8R^2(x_n-1)^2 -$$

$$- 16R^2y_{n+1}^2 + 2x_n^2(x_n-1)^2 + 4x_n^2y_{n+1}^2 + 4(x_n-1)^2y_{n+1}^2 .$$

Efectuînd reducerile, rezultă:

$$2x_n^2(x_n-1)^2 < 16R^4 + x_n^4 + (x_n-1)^4 - 8R^2x_n^2 - 8R^2(x_n-1)^2 - 16R^2y_{n+1}^2 \Rightarrow$$

$$\Rightarrow 8R^2[x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 - 2R^2] < x_n^4 + (x_n-1)^4 - 2x_n^2(x_n-1)^2 =$$

$$= [x_n^2 - (x_n-1)^2]^2 . \quad (3.7)$$

Dar, conform celei de-a 2-a inegalităţi (3.6), avem:

$$x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 - 2R^2 \geq 1$$

deoarece numărul din membrul stîng este întreg, impar şi strict pozitiv. Prin urmare, din (3.7) rezultă:

$$8R^2 < [x_n^2 - (x_n-1)^2]^2 = (2x_n-1)^2 \Rightarrow 2x_n-1 > 2\sqrt{2}R .$$

Am obţinut o contradicţie cu condiţia evidentă $x_n \leq R, n \geq 0$, ceea ce încheie demonstraţia. ■

Rezultatul obţinut permite înlocuirea criteriului (3.3) de alegere a punctului (x_{n+1}, y_{n+1}) cu criteriul echivalent:

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 \quad (3.8)$$

$$\Delta_n < 0 \Rightarrow \begin{cases} x_{n+1} = x_n \\ y_{n+1} = y_n+1 \end{cases}$$

$$\Delta_n > 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n+1 \end{cases}$$

(numerele Δ_n sînt întregi şi impare, deci nenule).

Se poate dezvolta un algoritm de trasare incrementală a unui cerc pe baza criteriului (3.8), calculînd cîte o valoare Δ_n pentru fiecare punct (x_n, y_n) generat. Vor fi efectuate adunări, scăderi şi înmulţiri de numere întregi.

Totuşi, numerele Δ_n se pot calcula mai uşor pe baza unei formule de recurenţă, în care nu mai apar operaţii de înmulţire. Pentru a găsi această formulă, să evaluăm diferenţa $\Delta_{n+1} - \Delta_n$, pe care o vom exprima în funcţie de coordonatele noului punct (x_{n+1}, y_{n+1}):

$$\Delta_{n+1} - \Delta_n = [x_{n+1}^2 + (x_{n+1}-1)^2 + 2(y_{n+1}+1)^2 - 2R^2] - \quad (3.9)$$

$$- [x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2] =$$

$$= x_{n+1}^2 + (x_{n+1}-1)^2 + 2(y_{n+1}+1)^2 - x_n^2 - (x_n-1)^2 - 2y_{n+1}^2 =$$

$$= x_{n+1}^2 + (x_{n+1}-1)^2 - x_n^2 - (x_n-1)^2 + 4y_{n+1}+2 .$$

Dacă $\Delta_n < 0$, atunci $x_{n+1} = x_n$ şi avem $\Delta_{n+1} - \Delta_n = 4y_{n+1}+2$.
Dacă $\Delta_n > 0$, atunci $x_{n+1} = x_n-1$ şi din (3.9) rezultă:

$$\Delta_{n+1} - \Delta_n = x_{n+1}^2 + (x_{n+1}-1)^2 - (x_{n+1}+1)^2 - x_{n+1}^2 + 4y_{n+1}+2 =$$

$$= -4x_{n+1} + 4y_{n+1}+2 .$$

Relaţiile de recurenţă obţinute permit calcularea mai simplă (fără înmulţiri) a noii valori Δ_{n+1} pornind de la Δ_n. Sînt necesare operaţii de adunare şi scădere de numere întregi şi de deplasare binară la stînga, pentru efectuarea înmulţirilor cu 4. Scriind împreună criteriul (3.8) şi relaţiile de recurenţă obţinute pentru Δ_n, obţinem formulele pe care se bazează algoritmul de trasare incrementală a cercului, în primul octant:

$$\Delta_n < 0 \Rightarrow \begin{cases} x_{n+1} = x_n \\ y_{n+1} = y_n+1 \\ \Delta_{n+1} = \Delta_n + 4y_{n+1} + 2 \end{cases} \tag{3.10}$$

$$\Delta_n > 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n+1 \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} + 4y_{n+1} + 2 . \end{cases}$$

Cele 3 numere (x_n, y_n, Δ_n) definesc starea curentă a algoritmului pentru fiecare punct generat. Pentru aceste numere vor fi alocate variabilele de stare ale algoritmului. Valorile de iniţializare pentru cele 3 variabile de stare se obţin uşor, dacă ţinem seama că primul punct generat în octantul 1 este $(R, 0)$:

$$x_0 = R \tag{3.11}$$

$$y_0 = 0$$

$$\Delta_0 = x_0^2 + (x_0-1)^2 + 2(y_0+1)^2 - 2R^2 = R^2 + (R-1)^2 + 2 - 2R^2 =$$

$$= 3 - 2R .$$

Cu aceste pregătiri, putem formula algoritmul de principiu pentru generarea incrementală a unui arc de cerc în primul octant. Numărul de puncte generate în octantul 1, precum şi momentul intrării în octantul 2 (care coincide cu traversarea primei bisectoare) nu sînt precizate; aceste chestiuni necesită o analiză specială care este expusă în secţiunile 8.3.3 şi 8.3.6.

Algoritmul C1

Generarea incrementală a unui arc de cerc de centru (x_c, y_c) și rază R, în primul octant.

Pasul 1: Inițializări: $n \leftarrow 0, x_0 \leftarrow R, y_0 \leftarrow 0, \Delta_0 \leftarrow 3 - 2R$.

Pasul 2: *call plot* (x_c+x_n, y_c+y_n).

Pasul 3: Dacă $\Delta_n > 0$, salt la Pasul 5.

Pasul 4: $x_{n+1} \leftarrow x_n$;
$y_{n+1} \leftarrow y_n + 1$;
$\Delta_{n+1} \leftarrow \Delta_n + 4y_{n+1} + 2$;
$n \leftarrow n + 1$;
salt la Pasul 2.

Pasul 5: $x_{n+1} \leftarrow x_n - 1$;
$y_{n+1} \leftarrow y_n + 1$;
$\Delta_{n+1} \leftarrow \Delta_n - 4x_{n+1} + 4y_{n+1} + 2$;
$n \leftarrow n + 1$;
salt la Pasul 2.

Algoritmul de principiu C1 nu utilizează variabile de adresare fizică în spațiul de afișare, dar se ține seama că argumentele subrutinei *plot* sînt coordonate absolute, iar (x_n, y_n) sînt coordonate relative la centrul cercului.

Înainte de a trece la analiza trasării în ceilalți octanți, să demonstrăm două relații care ne vor fi utile în continuare.

Teoremă. *Pentru fiecare punct* (x_n, y_n) *generat în octantul 1, variabilele de stare* x_n, y_n, Δ_n *verifică următoarele inegalități:*

$$4y_n + 2 - 4x_n < \Delta_n < 4y_n + 2. \qquad (3.12)$$

Demonstrație.

Să notăm:

$$S_n = \Delta_n + 4x_n - 4y_n - 2$$

$$T_n = \Delta_n - 4y_n - 2, \; n = 0, 1, 2, \ldots$$

și să demonstrăm prin inducție după n că $S_n > 0$ și $T_n < 0$, $\forall n \geq 0$. La inițializare:

$$S_0 = \Delta_0 + 4x_0 - 4y_0 - 2 = 3 - 2R + 4R - 2 = 2R + 1 > 0$$

$$T_0 = \Delta_0 - 4y_0 - 2 = 3 - 2R - 2 = 1 - 2R < 0$$

conform cu (3.11). Să presupunem că $S_n > 0$, $T_n < 0$ şi să evaluăm S_{n+1} şi T_{n+1}. Din expresiile (3.10), în cazul $\Delta_n < 0$, deducem:

$$\Delta_{n+1} = \Delta_n + 4y_{n+1} + 2 \Rightarrow$$

$$S_{n+1} = \Delta_{n+1} + 4x_{n+1} - 4y_{n+1} - 2 =$$

$$= (\Delta_n + 4y_{n+1} + 2) + 4x_{n+1} - 4y_{n+1} - 2 =$$

$$= \Delta_n + 4x_{n+1} = \Delta_n + 4x_n = S_n + 4y_n + 2 > 0;$$

$$T_{n+1} = \Delta_{n+1} - 4y_{n+1} - 2 = (\Delta_n + 4y_{n+1} + 2) - 4y_{n+1} - 2 = \Delta_n < 0.$$

În cazul $\Delta_n > 0$ deducem:

$$\Delta_{n+1} = \Delta_n - 4x_{n+1} + 4y_{n+1} + 2 \Rightarrow$$

$$S_{n+1} = \Delta_{n+1} + 4x_{n+1} - 4y_{n+1} - 2 = \Delta_n > 0;$$

$$T_{n+1} = \Delta_{n+1} - 4y_{n+1} - 2 = \Delta_n - 4x_{n+1} = T_n + 4y_n + 2 - 4x_{n+1} =$$

$$= T_n - 4x_{n+1} + 4y_{n+1} - 2 < 0,$$

ultima relaţie rezultînd din faptul că $x_n \geq y_n$ pentru orice punct generat în octantul 1. ■

8.3.2. Formule de recurenţă pentru octanţii 2, 3 şi 4

Odată rezolvată problema generării cercului în primul octant, pentru trasarea în ceilalţi octanţi se poate proceda prin simetrie (fig. 8–9).

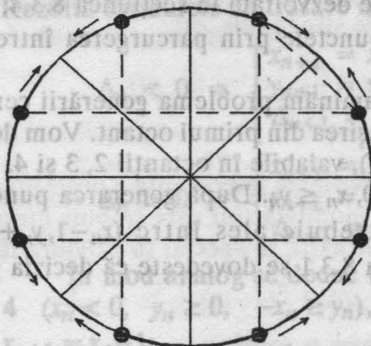


Figura 8–9.

Generarea prin simetrie a punctelor cercului din octanţii 2, 3, ... 8.

Ideea este de a utiliza concomitent 8 perechi de variabile de adresare fizică, cîte două coordonate x, y pentru fiecare octant; pentru fiecare punct generat în octantul 1, se generează şi simetricele lui în ceilalţi octanţi, şi pentru fiecare deplasare elementară în octantul 1, calculată pe baza relaţiilor (3.10), se efectuează cîte o deplasare simetrică în octanţii 2, 3, ... 8.

Această abordare a problemei prezintă în practică două dezavantaje importante:

- Sînt necesare prea multe variabile de adresare fizică; în mod normal, setul de registre generale ale procesorului este insuficient pentru toate variabilele. Prin urmare, unele din variabile trebuie stocate în memorie; timpul pierdut pentru accesele repetate la variabilele din memorie nu este compensat de cîştigul realizat prin faptul că recurenţele de tipul (3.10) sînt calculate numai în octantul 1.
- Există situaţii cînd ordinea generării punctelor cercului are importanţă. De exemplu, dacă trasarea trebuie să se efectueze nu cu linie continuă, ci cu linie întreruptă corespunzînd unui pattern (configuraţie liniară de biţi) predefinit, generarea prin simetrie a punctelor nu permite repetarea circulară a pattern-ului de-a lungul întregului cerc.

În funcţie de posibilităţile procesorului şi de necesităţile aplicaţiei, alte variante de trasare a cercului pot fi utilizate, cu sau fără generarea prin simetrie a mai multor puncte simultan:

- se calculează pe rînd poziţiile punctelor în octanţii 1 şi 2 şi se generează cîte 4 puncte simultan: punctul din primul cadran împreună cu simetricele lui faţă de axele *Ox*, *Oy* şi faţă de origine (centrul cercului);
- se calculează pe rînd poziţiile punctelor în octanţii 1, 2, 3 şi 4 şi se generează simultan cîte 2 puncte simetrice faţă de axa *Ox*;
- se calculează poziţiile punctelor succesive în octanţii 1, 2, 3 şi 4 şi se generează simultan cîte 2 puncte simetrice faţă de centrul cercului. Această variantă este dezvoltată în secţiunea 8.3.4;
- se generează pe rînd toate punctele prin parcurgerea întregului cerc într-un singur sens.

Prin urmare, este necesar să examinăm problema generării cercului în sensul trigonometric direct, după ieşirea din primul octant. Vom deduce relaţii de recurenţă analoage cu (3.10), valabile în octanţii 2, 3 şi 4.

În octantul 2 avem $x_n \geq 0, y_n > 0, x_n \leq y_n$. După generarea punctului (x_n, y_n) următorul punct generat trebuie ales între (x_n-1, y_n+1) şi (x_n-1, y_n). Exact la fel ca în secţiunea 8.3.1 se dovedeşte că decizia poate fi luată pe baza semnului cantităţii:

$$\Delta_n = [(x_n-1)^2 + (y_n+1)^2 - R^2] - [R^2 - (x_n-1)^2 - y_n^2] = \\ = 2(x_n-1)^2 + (y_n+1)^2 + y_n^2 - 2R^2 . \tag{3.13}$$

Dacă $\Delta_n < 0$ se alege $(x_{n+1}, y_{n+1}) = (x_n-1, y_n+1)$, iar dacă $\Delta_n > 0$ se alege $(x_{n+1}, y_{n+1}) = (x_n-1, y_n)$. Calculînd:

$$\Delta_{n+1} - \Delta_n = [2(x_{n+1}-1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2] -$$

$$- [2(x_n-1)^2 + (y_n+1)^2 + y_n^2 - 2R^2] =$$

$$= 2(x_{n+1}-1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2x_{n+1}^2 - (y_n+1)^2 - y_n^2 =$$

$$= -4x_{n+1} + 2 + (y_{n+1}+1)^2 + y_{n+1}^2 - (y_n+1)^2 - y_n^2$$

deducem formulele de trasare incrementală a cercului în octantul 2:

$$\Delta_n < 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n+1 \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} + 4y_{n+1} + 2 \end{cases} \tag{3.14}$$

$$\Delta_n > 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} + 2\,. \end{cases}$$

În octantul 3, avem $x_n \leq 0$, $y_n > 0$, $-x_n \leq y_n$. După generarea punctului (x_n, y_n), punctele candidat sînt (x_n-1, y_n) și (x_n-1, y_n-1). În acest caz, obținem:

$$\Delta_n = [(x_n-1)^2 + y_n^2 - R^2] - [R^2 - (x_n-1)^2 - (y_n-1)^2] = \tag{3.15}$$

$$= 2(x_n-1)^2 + y_n^2 + (y_n-1)^2 - 2R^2$$

$$\Delta_{n+1} - \Delta_n = 2(x_{n+1}-1)^2 + y_{n+1}^2 + (y_{n+1}-1)^2 - 2(x_n-1)^2 - y_n^2 - (y_n-1)^2 =$$

$$= -4x_{n+1} + 2 + y_{n+1}^2 + (y_{n+1}-1)^2 - y_n^2 - (y_n-1)^2.$$

Rezultă formulele de trasare incrementală a cercului în octantul 3:

$$\Delta_n < 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} + 2 \end{cases} \tag{3.16}$$

$$\Delta_n > 0 \Rightarrow \begin{cases} x_{n+1} = x_n-1 \\ y_{n+1} = y_n-1 \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} - 4y_{n+1} + 2\,. \end{cases}$$

În mod analog se deduc formulele de recurență valabile în octantul 4 ($x_n < 0$, $y_n \geq 0$, $-x_n \geq y_n$), în care $y_{n+1} = y_n-1$ și $x_{n+1} = x_n$ sau $x_{n+1} = x_n-1$:

$$\Delta_n = [(x_n-1)^2 + (y_n-1)^2 - R^2] - [R^2 - x_n^2 - (y_n-1)^2] = \tag{3.17}$$

$$= (x_n-1)^2 + x_n^2 + 2(y_n-1)^2 - 2R^2$$

$$\Delta_{n+1} - \Delta_n = (x_{n+1}-1)^2 + x_{n+1}^2 + 2(y_{n+1}-1)^2 - (x_n-1)^2 - x_n^2 - 2(y_n-1)^2 =$$

$$= -4y_{n+1} + 2 + (x_{n+1}-1)^2 + x_{n+1}^2 - (x_n-1)^2 - x_n^2$$

$$\Delta_n < 0 \Rightarrow \begin{cases} x_{n+1} = x_n - 1 \\ y_{n+1} = y_n - 1 \\ \Delta_{n+1} = \Delta_n - 4x_{n+1} - 4y_{n+1} + 2 \end{cases} \qquad (3.18)$$

$$\Delta_n > 0 \Rightarrow \begin{cases} x_{n+1} = x_n \\ y_{n+1} = y_n - 1 \\ \Delta_{n+1} = \Delta_n - 4y_{n+1} + 2\,. \end{cases}$$

După cum s-a arătat mai înainte, în cazul trasării conturului circular cu linie întreruptă conform unui pattern predefinit, este necesară generarea întregului cerc într-un singur sens. Totuși, nu este necesară deducerea unor formule de recurență speciale pentru octanții 5, 6, 7 și 8. Judecînd prin simetrie față de centrul cercului, ajungem la concluzia că se pot utiliza formulele deduse pînă acum, dacă se respectă următoarele condiții:

- Pentru trasarea în octanții 5, 6, 7 și 8 se vor utiliza formulele de recurență valabile, respectiv, pentru octanții 1, 2, 3 și 4.
- Variabilele de stare (x_n, y_n, Δ_n) se reinițializează la intrarea în octantul 5, la fel ca în octantul 1: $x_0 = R$, $y_0 = 0$, $\Delta_0 = 3-2R$.
- Pentru trecerea de la un octant la altul, se utilizează formulele valabile pentru trecerile între octanții corespunzători din semicercul superior, deduse în secțiunea următoare 8.3.3.
- Toate deplasările elementare efectuate de algoritm pentru generarea semicercului superior, se înlocuiesc prin deplasări elementare în sens opus pentru trasarea semicercului inferior: deplasare în jos în loc de în sus, spre dreapta-jos în loc de stînga-sus, etc.

8.3.3. Chestiuni de frontieră

În această secțiune vom analiza problema importantă a posibilităților de trecere dintr-un octant în altul și a modificărilor care trebuie efectuate asupra variabilelor de stare în aceste situații, în special în privința lui Δ_n.

8.3.3.1. Traversarea primei bisectoare a axelor

Traversarea primei bisectoare a axelor poate avea loc în trei moduri, analizate mai jos. În toate cazurile, notăm cu $A = (x_n, y_n)$ ultimul punct generat în interiorul octantului 1 și cu $B = (x_{n+1}, y_{n+1})$ primul punct din octantul 2 sau de pe bisectoare.

8.3.3.1.1. Intrarea în octantul 2 prin generarea unui punct pe bisectoare.

În situaţia din fig. 8–10, ultima deplasare din octantul 1 este diagonală şi primul punct generat în octantul 2 este pe bisectoare.

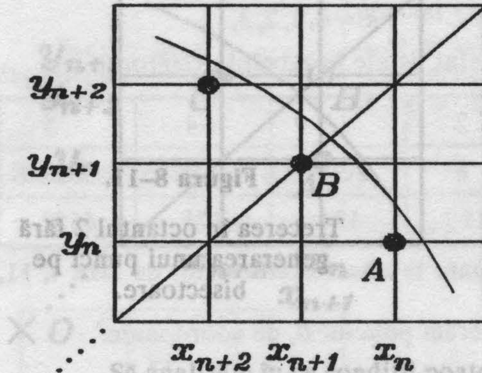


Figura 8–10.

Intrarea în octantul 2 prin generarea unui punct pe bisectoare.

Prin urmare, acest caz survine atunci cînd coordonatele punctului care trebuie generat devin egale: $x_{n+1} = y_{n+1}$. Pentru a iniţializa variabila Δ_{n+1} corespunzătoare acestui punct, în octantul 2, avem de calculat:

$$\Delta_{n+1} = 2(x_{n+1}-1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 = \qquad (3.19)$$

$$= 2(x_{n+1}-1)^2 + (x_{n+1}+1)^2 + x_{n+1}^2 - 2R^2 = 4x_{n+1}^2 - 2x_{n+1} + 3 - 2R^2$$

conform cu formula (3.13). Numărul x_{n+1} este cunoscut, deoarece punctul A, de coordonate:

$$(x_n, y_n) = (x_{n+1}+1, y_{n+1}-1) = (x_{n+1}+1, x_{n+1}-1)$$

a fost deja generat în octantul 1. Prin urmare, este posibil să iniţializăm Δ_{n+1} folosind (3.19). Totuşi, această formulă este relativ complicată şi este de dorit să fie evitate înmulţirile. Să examinăm valoarea Δ_n, corespunzătoare punctului $A = (x_n, y_n)$ din octantul 1. În conformitate cu (3.8) avem:

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 = \qquad (3.20)$$

$$= (x_{n+1}+1)^2 + x_{n+1}^2 + 2x_{n+1}^2 - 2R^2 = 4x_{n+1}^2 + 2x_{n+1} + 1 - 2R^2 .$$

Comparînd (3.19) cu (3.20), deducem formulele de intrare în octantul 2, corespunzătoare cazului din fig. 8–15:

$$\begin{matrix} x_{n+1} = y_{n+1} \\ \text{(după deplasare diagonală)} \end{matrix} \Rightarrow \Delta_{n+1} = \Delta_n - 4x_{n+1} + 2. \qquad (3.21)$$

8.3.3.1.2. Trecerea în octantul 2 fără generarea unui punct pe bisectoare.

În situaţia exemplificată în fig. 8–11, ultima deplasare din octantul 1 este diagonală şi primul punct generat în octantul 2 este în interiorul acestuia.

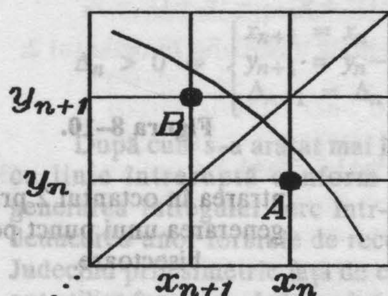


Figura 8–11.

Trecerea în octantul 2 fără generarea unui punct pe bisectoare.

Scriind şi în acest caz coordonatele ambelor puncte în funcţie de x_{n+1}, obţinem:

$$(x_n,y_n) = (x_{n+1}+1,x_{n+1}),\quad (x_{n+1},y_{n+1}) = (x_{n+1},x_{n+1}+1).$$

Prin urmare, traversarea bisectoarei ca în fig. 8–11 are loc cînd se îndeplineşte condiţia $x_{n+1} < y_{n+1}$ (în interiorul octantului 1 avem întotdeauna $x_n > y_n$).

În primul octant avem:

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 = \tag{3.22}$$

$$= (x_{n+1}+1)^2 + x_{n+1}^2 + 2(x_{n+1}+1)^2 - 2R^2 = 4x_{n+1}^2 + 6x_{n+1} + 3 - 2R^2.$$

În al doilea octant avem:

$$\Delta_{n+1} = 2(x_{n+1}-1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 = \tag{3.23}$$

$$= 2(x_{n+1}-1)^2 + (x_{n+1}+2)^2 + (x_{n+1}+1)^2 - 2R^2 =$$

$$= 4x_{n+1}^2 + 2x_{n+1} + 7 - 2R^2.$$

Din (3.22) şi (3.23) deducem formulele de trecere în octantul 2, corespunzătoare traversării primei bisectoare ca în fig. 8–11:

$$x_{n+1} < y_{n+1} \Rightarrow \Delta_{n+1} = \Delta_n - 4x_{n+1} + 4. \tag{3.24}$$

8.3.3.1.3. Intrarea în octantul 2 printr-o deplasare pe verticală.

Deşi pare surprinzător la prima vedere, este posibil ca ultima deplasare în octantul 1 să fie pe verticală, ajungîndu-se la un punct B pe bisectoare (fig. 8-12).

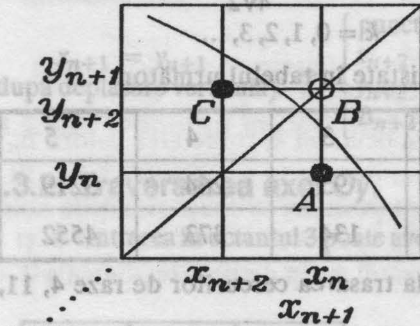


Figura 8-12.

Intrarea în octantul 2 printr-o deplasare pe verticală.

Să analizăm în ce condiţii poate apare această situaţie. Coordonatele punctului A sînt $(x_n, y_n) = (x_n, x_n-1)$. Deoarece ultima deplasare din octantul 1 este pe verticală, înseamnă că în punctul A avem $\Delta_n < 0$ (în conformitate cu (3.10)), prin urmare:

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 = \tag{3.25}$$

$$= x_n^2 + (x_n-1)^2 + 2x_n^2 - 2R^2 = 4x_n^2 - 2x_n + 1 - 2R^2 < 0 .$$

Aplicînd prima inegalitate (3.12) pentru punctul A din fig. 8-12, obţinem:

$$\Delta_n > 4y_n + 2 - 4x_n = 4(x_n-1) + 2 - 4x_n = -2 . \tag{3.26}$$

Ţinînd seama de faptul că Δ_n este un număr întreg, inegalităţile simultane (3.25) şi (3.26) implică, în punctul A:

$$\Delta_n = -1 \Rightarrow 4x_n^2 - 2x_n + 2 - 2R^2 = 0; \tag{3.27}$$

$$2x_n^2 - x_n + 1 - R^2 = 0 .$$

Prin urmare, situaţia exemplificată în fig. 8-12 poate apare numai dacă ecuaţia diofantică:

$$2x^2 - x + 1 - R^2 = 0 \tag{3.28}$$

are soluţii în numere întregi. Ar fi avantajos ca această ecuaţie să *nu* aibă soluţii, deoarece în acest caz, în algoritmul de trasare, testul de ieşire din octantul 1 nu ar fi necesar după o deplasare pe verticală (ci numai după o deplasare diagonală, ca în fig. 8-10 şi fig. 8-11). Totuşi, se dovedeşte că ecuaţia (3.28) *are soluţii* în numere întregi, şi anume o infinitate; forma generală a soluţiilor este:

$$x_k = \frac{(3+2\sqrt{2})^k - (3-2\sqrt{2})^k}{2\sqrt{2}} + (-1)^{k+1}\frac{(3+2\sqrt{2})^k + (3-2\sqrt{2})^k}{8} + \frac{1}{4}$$

$$R_k = \frac{(3+2\sqrt{2})^k + (3-2\sqrt{2})^k}{2} + (-1)^{k+1}\frac{(3+2\sqrt{2})^k - (3-2\sqrt{2})^k}{4\sqrt{2}},$$

$$k = 0, 1, 2, 3, \ldots$$

Primele 6 perechi de soluţii sînt listate în tabelul următor.

$k =$	0	1	2	3	4	5
$x_k =$	0	3	8	95	264	3219
$R_k =$	1	4	11	134	373	4552

Deci situaţia din fig. 8–12 apare la trasarea cercurilor de raze 4, 11, 134, 373, etc.

Să presupunem acum că generăm punctul B, de coordonate:

$$(x_{n+1}, y_{n+1}) = (x_n, y_n+1) = (x_n, x_n)$$

conform evoluţiei algoritmului în octantul 1. Următoarea deplasare elementară se calculează în octantul 2, pe baza formulelor (3.13) şi (3.14) aplicate punctului (x_{n+1}, y_{n+1}):

$$\Delta_{n+1} = 2(x_{n+1}-1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 =$$

$$= 2(x_n-1)^2 + (x_n+1)^2 + x_n^2 - 2R^2 = 4x_n^2 - 2x_n + 3 - 2R^2 .$$

Din (3.27) deducem $\Delta_{n+1} = 1 > 0$, deci, cum era de aşteptat de altfel, prima deplasare în octantul 2 este orizontală şi următorul punct generat trebuie să fie C (fig. 8–12). Dar succesiunea de 3 puncte A, B, C nu îndeplineşte condiţia de conexiune discretă precizată în secţiunea 8.1; este clar că punctul B nu este necesar.

Prin urmare, după generarea punctului $A = (x_n, x_n-1)$, următorul punct generat de algoritm trebuie să fie C, ale cărui coordonate le notăm (x_{n+2}, y_{n+2}). Din fig. 8–12 rezultă:

$$(x_{n+2}, y_{n+2}) = (x_n-1, y_n+1) = (x_n-1, x_n) = (x_{n+2}, x_{n+2}+1) .$$

În octantul 2, pentru punctul C putem scrie:

$$\Delta_{n+2} = 2(x_{n+2}-1)^2 + (y_{n+2}+1)^2 + y_{n+2}^2 - 2R^2 =$$

$$= 2(x_n-2)^2 + (x_n+1)^2 + x_n^2 - 2R^2 =$$

$$= 4x_n^2 - 6x_n + 9 - 2R^2 = \Delta_n - 4x_n + 8 = -1 - 4x_{n+2} + 4$$

(am folosit relaţiile (3.25) şi (3.27)).

În concluzie, dacă după o deplasare pe verticală algoritmul detectează atingerea primei bisectoare ($x_{n+1} = y_{n+1}$), atunci punctul (x_{n+1}, y_{n+1}) nu trebuie generat și trebuie să se treacă direct la punctul (x_{n+2}, y_{n+2}) în interiorul octantului 2, prin formulele de trecere:

$$\begin{matrix} x_{n+1} = y_{n+1} \\ \text{(după deplasare verticală)} \end{matrix} \Rightarrow \begin{cases} \text{punctul } (x_{n+1}, y_{n+1}) \text{ nu este generat} \\ x_{n+2} = x_{n+1} - 1 = x_n - 1 \\ y_{n+2} = y_{n+1} = y_n + 1 \\ \Delta_{n+2} = \Delta_n - 4x_{n+2} + 4 = -1 - 4x_{n+2} + 4 . \end{cases} \quad (3.29)$$

8.3.3.2. Traversarea axei *Oy*.

Intrarea în octantul 3 poate avea loc într–un singur mod, descris în fig. 8–13.

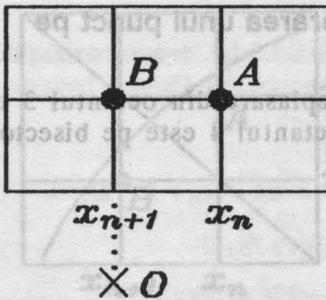


Figura 8–13.

Traversarea axei *Oy*.

Ultimul punct *A* generat în octantul 2 are coordonatele $(x_n, y_n) = (1, R)$ pentru toate cercurile de rază $R \geq 2$. Conform formulei (3.13), în punctul *A* putem scrie:

$$\Delta_n = 2(x_n - 1)^2 + (y_n + 1)^2 + y_n^2 - 2R^2 =$$
$$= 2(1-1)^2 + (R+1)^2 + R^2 - 2R^2 = 2R + 1 .$$

Primul punct generat în octantul 3 este *B* de coordonate $(x_{n+1}, y_{n+1}) = (0, R)$. Din relația (3.15) deducem:

$$\Delta_{n+1} = 2(x_{n+1} - 1)^2 + y_{n+1}^2 + (y_{n+1} - 1)^2 - 2R^2 =$$
$$= 2(0-1)^2 + R^2 + (R-1)^2 - 2R^2 =$$
$$= 3 - 2R .$$

Evident, intrarea în octantul 3 este detectată prin condiția $x_{n+1} = 0$ pentru punctul care trebuie generat. Prin urmare, formula de trecere din octantul 2 în octantul 3 este:

$$x_{n+1} = 0 \Rightarrow \Delta_{n+1} = 3 - 2R = 4 - \Delta_n . \quad (3.30)$$

Ultima deplasare elementară în octantul 2 nu poate fi diagonală, pentru $R \geq 2$; într–adevăr, punctul $(1, R-1)$, de unde porneşte această deplasare diagonală, nu este generat de algoritm deoarece este mai depărtat de cercul ideal decît punctul $(1, R)$. Prin urmare, testul de ieşire din octantul 2 $(x_{n+1} : 0)$ nu este necesar după o deplasare diagonală, ci numai după o deplasare orizontală.

8.3.3.3. Trecerea din octantul 3 în octantul 4

Pentru trecerea din octantul 3 în octantul 4, corectarea valorii Δ_n se face prin relaţii asemănătoare cu cele deduse în secţiunea 8.3.3.1. Şi în acest caz trebuie analizate trei situaţii posibile; în toate situaţiile notăm cu $A = (x_n, y_n)$ ultimul punct generat în interiorul octantului 3 şi cu $B = (x_{n+1}, y_{n+1})$ primul punct din octantul 4 sau de pe bisectoarea a 2–a.

8.3.3.3.1. Intrarea în octantul 4 prin generarea unui punct pe bisectoare

În situaţia din fig. 8–14, ultima deplasare din octantul 3 este diagonală şi primul punct generat în octantul 4 este pe bisectoare $(-x_{n+1} = y_{n+1})$.

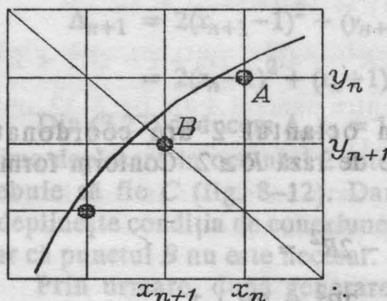


Figura 8–14.
Intrarea în octantul 4 prin generarea unui punct pe bisectoare.

Pentru punctul A în octantul 3, din relaţia (3.15) obţinem:

$$\Delta_n = 2(x_n-1)^2 + y_n^2 + (y_n-1)^2 - 2R^2 =$$
$$= 2x_{n+1}^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 =$$
$$= 2y_{n+1}^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 = 4y_{n+1}^2 + 2y_{n+1} + 1 - 2R^2 .$$

Pentru punctul B în octantul 4, din relaţia (3.17) rezultă:

$$\Delta_{n+1} = (x_{n+1}-1)^2 + x_{n+1}^2 + 2(y_{n+1}-1)^2 - 2R^2 =$$

$$= (-y_{n+1}-1)^2 + y_{n+1}^2 + 2(y_{n+1}-1)^2 - 2R^2 =$$
$$= 4y_{n+1}^2 - 2y_{n+1} + 3 - 2R^2.$$

Prin urmare, formulele de intrare în octantul 4, în această situație, sînt următoarele:

$$\begin{array}{l} -x_{n+1} = y_{n+1} \\ \text{(după deplasare diagonală)} \end{array} \Rightarrow \Delta_{n+1} = \Delta_n - 4y_{n+1} + 2. \quad (3.31)$$

8.3.3.3.2. Trecerea în octantul 4 fără generarea unui punct pe bisectoare

Altă posibilitate este exemplificată în fig. 8–15: ultima deplasare din octantul 3 este diagonală și primul punct generat în octantul 4 este în interiorul acestuia ($-x_{n+1} = y_{n+1}+1 > y_{n+1}$).

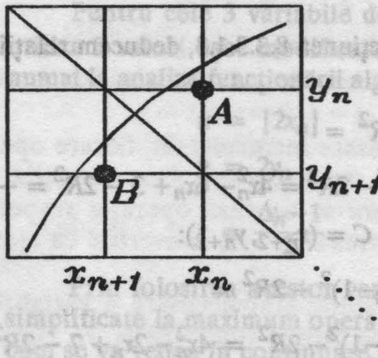


Figura 8–15.

Trecerea în octantul 4 fără generarea unui punct pe bisectoare.

Din relațiile (3.15) și (3.17) deducem succesiv:

$$\Delta_n = 2(x_n-1)^2 + y_n^2 + (y_n-1)^2 - 2R^2 =$$
$$= 2x_{n+1}^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 =$$
$$= 2(y_{n+1}+1)^2 + (y_{n+1}+1)^2 + y_{n+1}^2 - 2R^2 =$$
$$= 4y_{n+1}^2 + 6y_{n+1} + 3 - 2R^2;$$

$$\Delta_{n+1} = (x_{n+1}-1)^2 + x_{n+1}^2 + 2(y_{n+1}-1)^2 - 2R^2 =$$
$$= (-y_{n+1}-2)^2 + (-y_{n+1}-1)^2 + 2(y_{n+1}-1)^2 - 2R^2 =$$
$$= 4y_{n+1}^2 + 2y_{n+1} + 7 - 2R^2.$$

În acest caz, formulele de trecere din octantul 3 în octantul 4 sînt:

$$-x_{n+1} > y_{n+1} \Rightarrow \Delta_{n+1} = \Delta_n - 4y_{n+1} + 4. \quad (3.32)$$

8.3.3.3.3. Intrarea în octantul 4 printr–o deplasare orizontală

La fel ca în primul cadran, este posibil ca ultima deplasare din octantul 3 să fie orizontală, ajungîndu–se la un punct B pe bisectoare (fig. 8–16).

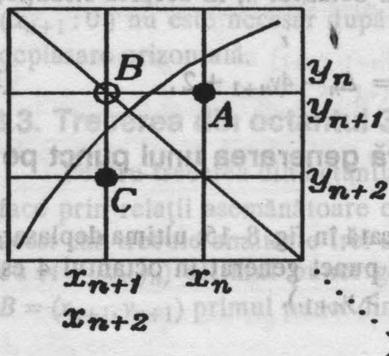


Figura 8–16.

Intrarea în octantul 4 printr–o deplasare pe orizontală.

În acest caz, procedînd analog ca în secţiunea 8.3.3.1.3, deducem relaţiile:

- în octantul 3:

$$\Delta_n = 2(x_n-1)^2 + y_n^2 + (y_n-1)^2 - 2R^2 =$$
$$= 2(x_n-1)^2 + (-x_n+1)^2 + (-x_n)^2 - 2R^2 = 4x_n^2 - 6x_n + 3 - 2R^2 = -1.$$

- în octantul 4, pentru punctul $C = (x_{n+2}, y_{n+2})$:

$$\Delta_{n+2} = (x_{n+2}-1)^2 + x_{n+2}^2 + 2(y_{n+2}-1)^2 - 2R^2 =$$
$$= (x_n-2)^2 + (x_n-1)^2 + 2(-x_n-1)^2 - 2R^2 = 4x_n^2 - 2x_n + 7 - 2R^2 =$$
$$= \Delta_n + 4x_n + 4 = -1 - 4y_{n+2} + 4\,.$$

Dacă în octantul 3 algoritmul detectează atingerea bisectoarei a doua ($-x_{n+1} = y_{n+1}$) după o deplasare orizontală, atunci punctul (x_{n+1}, y_{n+1}) nu trebuie generat şi trebuie să se treacă direct la punctul (x_{n+2}, y_{n+2}) în interiorul octantului 4, prin formulele de trecere:

$$\begin{array}{c} -x_{n+1} = y_{n+1} \\ \text{(după deplasare orizontală)} \end{array} \Rightarrow \begin{cases} \text{punctul } (x_{n+1}, y_{n+1}) \text{ nu este generat} \\ x_{n+2} = x_{n+1} = x_n-1 \\ y_{n+2} = y_{n+1}-1 = y_n-1 \\ \Delta_{n+2} = \Delta_n - 4y_{n+2} + 4 = -1 - 4y_{n+2} + 4\,. \end{cases} \quad (3.33)$$

8.3.4. Descrierea algoritmului de generare incrementală a cercurilor

Vom prezenta în continuare algoritmul de trasare incrementală a cercurilor, în varianta generării a două puncte simultan, prin simetrie faţă

de centrul cercului (fig. 8–17). Algoritmul se bazează pe relaţiile de recurenţă deduse în secţiunile anterioare.

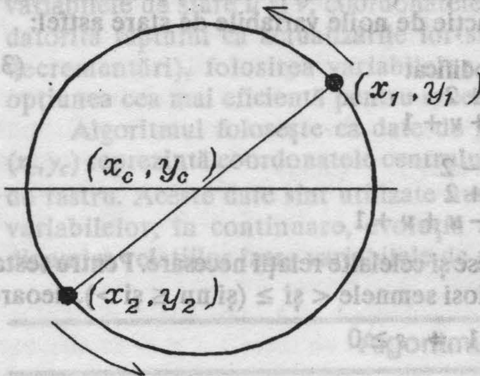


Figura 8–17.

Trasarea cercului prin generarea simultană a două puncte diametral opuse.

Pentru cele 3 variabile de stare, în locul mărimilor (x_n, y_n, Δ_n) vom utiliza 3 valori uşor modificate. Vom renunţa la indicele n, care a folosit numai la analiza funcţionării algoritmului, şi vom defini variabilele de stare:

$$u = |2x_n| \qquad (3.34)$$
$$v = 2y_n$$
$$s = \frac{\Delta_n - 1}{2}.$$

Prin folosirea acestor variabile, care sînt tot numere întregi, vor fi simplificate la maximum operaţiile aritmetice efectuate de algoritm, după cum se va vedea în continuare.

Introducerea variabilei s este posibilă deoarece mărimea Δ_n este un număr întreg impar, după cum rezultă din relaţiile de definiţie în primii 4 octanţi (3.8), (3.13), (3.15) şi (3.17).

Domeniul de variaţie al variabilelor u şi v rezultă imediat din relaţiile de definiţie (3.34): ambele iau valori pare pozitive, cuprinse între 0 şi $2R$. Pentru a evalua domeniul de variaţie al lui s, folosim inegalităţile (3.12). Din aceste două relaţii, precum şi din inegalităţile $R\sqrt{2}/2 < x_n \le R$, $0 \le y_n < R\sqrt{2}/2$ valabile în octantul 1, deducem:

$$4y_n + 2 - 4x_n < \Delta_n < 4y_n + 2 \Rightarrow -4R + 2 < \Delta_n < 4R + 2 \Rightarrow$$
$$\Rightarrow -2R < s \le 2R \qquad (3.35)$$

pentru toate punctele generate în octantul 1. De fapt, se poate arăta că inegalitatea (3.35) este general valabilă pentru toate punctele generate.

Prin urmare, variabilele de stare u, v şi s pot fi reprezentate în memorie pe 16 biţi, iar dacă raza cercurilor care se trasează este cel mult de 64 unităţi de rastru, 8 biţi sînt suficienţi pentru fiecare variabilă.

Prin introducerea variabilelor de stare u, v şi s definite ca în (3.34), relaţiile de recurenţă pe care se bazează funcţionarea algoritmului iau forma cea mai simplă. De exemplu, relaţiile (3.10) care definesc pasul incremental în octantul 1, se rescriu în funcţie de noile variabile de stare astfel:

$$s < 0 \Rightarrow \begin{cases} u \text{ nemodificat} \\ v \leftarrow v + 2 \\ s \leftarrow s + v + 1 \end{cases} \qquad (3.36)$$

$$s \geq 0 \Rightarrow \begin{cases} u \leftarrow u - 2 \\ v \leftarrow v + 2 \\ s \leftarrow s - u + v + 1 . \end{cases}$$

În mod analog se convertesc şi celelalte relaţii necesare. Pentru testarea semnului variabilei s se vor folosi semnele $<$ şi $\geq$ (şi nu $\leq$ şi $>$), deoarece:

$$\Delta_n > 0 \Rightarrow \Delta_n \geq 1 \Rightarrow s \geq 0$$

$$\Delta_n < 0 \Rightarrow \Delta_n \leq -1 \Rightarrow s \leq -1 \Rightarrow s < 0$$

conform definiţiei lui s din (3.34). Deoarece x_n este negativ în octanţii 3 şi 4, semnul variabilei u va fi inversat faţă de cel al lui x_n în formulele referitoare la această parte a cercului.

În descrierea algoritmului de trasare incrementală, fiecare operaţie în care este implicată o variabilă de stare u, v sau s este însoţită de o referinţă la formula teoretică pe care se bazează operaţia respectivă. Evident, formulele teoretice referite trebuie să fie convertite cu ajutorul ecuaţiilor de legătură (3.34).

În plus faţă de variabilele de stare, sînt necesare 4 variabile de adresare fizică în spaţiul de afişare, pe care le vom nota x_1, y_1, x_2, y_2 (fig. 8–17). (x_1,y_1) reprezintă adresa fizică a punctului curent generat în semicercul superior, iar (x_2,y_2) este adresa fizică a punctului diametral opus, generat concomitent. Fiecare deplasare elementară efectuată în cursul execuţiei algoritmului este reflectată prin modificarea corespunzătoare a variabilelor (x_1,y_1) şi (x_2,y_2), adică prin deplasarea fizică la un pixel alăturat în spaţiul de afişare. De exemplu, deplasarea spre stînga–sus în octantul 1 apare în algoritm sub forma:

$$u \leftarrow u - 2 ,\ v \leftarrow v + 2 ;$$
$$x_1 \leftarrow x_1 - 1 ,\ y_1 \leftarrow y_1 + 1 ;$$
$$x_2 \leftarrow x_2 + 1 ,\ y_2 \leftarrow y_2 - 1 .$$

Deplasarea elementară spre stînga în octantul 3 apare sub forma:

$$u \leftarrow u + 2 ;$$
$$x_1 \leftarrow x_1 - 1 ;$$
$$x_2 \leftarrow x_2 + 1$$

deoarece u evoluează în sens invers, datorită schimbării de semn.

De fapt, variabilele de adresare x_1, y_1, x_2, y_2 sînt oarecum redundante, în sensul că valorile lor pot fi calculate în orice moment în funcţie de variabilele de stare u şi v, coordonatele centrului cercului şi rază. Totuşi, datorită faptului că actualizările lor sînt foarte simple (incrementări şi decrementări), folosirea variabilelor de adresare fizică se impune ca opţiunea cea mai eficientă pentru execuţie.

Algoritmul foloseşte ca date de intrare 3 numere întregi pozitive: (x_c, y_c) reprezintă coordonatele centrului, iar R este raza cercului în unităţi de rastru. Aceste date sînt utilizate numai în Pasul 1 pentru iniţializarea variabilelor; în continuare, evoluţia algoritmului depinde exculsiv de dinamica relaţiilor între variabilele de stare.

Algoritmul C2

Generarea incrementală a unui cerc de centru (x_c, y_c) şi rază $R \geq 2$.

Pasul 1: Iniţializări: $x_1 \leftarrow x_c + R, y_1 \leftarrow y_c, x_2 \leftarrow x_c - R, y_2 \leftarrow y_c$;
$u \leftarrow 2R, v \leftarrow 0, s \leftarrow 1 - R$; (3.11)
call plot (x_1, y_1);
call plot (x_2, y_2).

Pasul 2: $y_1 \leftarrow y_1 + 1$;
$y_2 \leftarrow y_2 - 1$;
$v \leftarrow v + 2$; (3.10), (3.36)
dacă $u = v$, salt la Pasul 9. (3.29)

Pasul 3: *call plot* (x_1, y_1);
call plot (x_2, y_2);
$s \leftarrow s + v + 1$; (3.10), (3.36)
dacă $s < 0$, salt la Pasul 2. (3.10), (3.36)

Pasul 4: $x_1 \leftarrow x_1 - 1, y_1 \leftarrow y_1 + 1$;
$x_2 \leftarrow x_2 + 1, y_2 \leftarrow y_2 - 1$;
$u \leftarrow u - 2, v \leftarrow v + 2, s \leftarrow s - u$; (3.10), (3.36)
dacă $u > v$, salt la Pasul 3.

Pasul 5: Dacă $u = v$, salt la Pasul 8; (3.21)
$s \leftarrow s + 1$; (3.24)
salt la Pasul 8.

Pasul 6: $x_1 \leftarrow x_1 - 1, y_1 \leftarrow y_1 + 1$;
$x_2 \leftarrow x_2 + 1, y_2 \leftarrow y_2 - 1$;
$u \leftarrow u - 2, v \leftarrow v + 2, s \leftarrow s + v$. (3.14)

Pasul 7: $s \leftarrow s - u$. (3.14)

Pasul 8: *call plot* (x_1, y_1); *call plot* (x_2, y_2).

Pasul 9: $s \leftarrow s + 1$; (3.14)
dacă $s < 0$, salt la Pasul 6. (3.14)

Pasul 10: $x_1 \leftarrow x_1 - 1$;
$x_2 \leftarrow x_2 + 1$;
$u \leftarrow u - 2$; (3.14)
dacă $u \neq 0$, salt la Pasul 7. (3.30)

Pasul 11: *call plot* (x_1, y_1);
call plot (x_2, y_2);
$s \leftarrow 1 - s$. (3.30)

Pasul 12: $x_1 \leftarrow x_1 - 1$;
$x_2 \leftarrow x_2 + 1$;
$u \leftarrow u + 2$; (3.16)
dacă $u = v$, salt la Pasul 19. (3.33)

Pasul 13: *call plot* (x_1, y_1);
call plot (x_2, y_2);
$s \leftarrow s + u + 1$; (3.16)
dacă $s < 0$, salt la Pasul 12. (3.16)

Pasul 14: $x_1 \leftarrow x_1 - 1, y_1 \leftarrow y_1 - 1$;
$x_2 \leftarrow x_2 + 1, y_2 \leftarrow y_2 + 1$;
$u \leftarrow u + 2, v \leftarrow v - 2, s \leftarrow s - v$; (3.16)
dacă $u < v$, salt la Pasul 13.

Pasul 15: Dacă $u = v$, salt la Pasul 18; (3.31)
$s \leftarrow s + 1$; (3.32)
salt la Pasul 18.

Pasul 16: $x_1 \leftarrow x_1 - 1, y_1 \leftarrow y_1 - 1$;
$x_2 \leftarrow x_2 + 1, y_2 \leftarrow y_2 + 1$;
$u \leftarrow u + 2, v \leftarrow v - 2, s \leftarrow s + u$. (3.18)

Pasul 17: $s \leftarrow s - v$. (3.18)

Pasul 18: *call plot* (x_1, y_1); *call plot* (x_2, y_2).

Pasul 19: $s \leftarrow s + 1$; (3.18)
dacă $s < 0$, salt la Pasul 16. (3.18)

Pasul 20: $y_1 \leftarrow y_1 - 1$;
$y_2 \leftarrow y_2 + 1$;
$v \leftarrow v - 2$; (3.18)
dacă $v \neq 0$, salt la Pasul 17.

Pasul 21: Stop.

Algoritmul C2 funcţionează în 4 faze succesive, alcătuite respectiv din paşii 2–4, 6–10, 12–14 şi 16–20, care generează punctele cercului în octanţii 1+5, 2+6, 3+7 şi 4+8. Pasul 1 este de iniţializare, iar în paşii 5, 11 şi 15 se execută operaţii de legătură între faze.

În comparaţie cu algoritmul de principiu C1, algoritmul C2 prezintă cîteva modificări în ceea ce priveşte ordinea operaţiilor, menite să simplifice şi să mărească eficienţa algoritmului. Astfel, punctele de intersecţie ale cercului cu axele sînt generate în afara buclelor principale de execuţie: punctele $(R, 0)$ şi $(-R, 0)$ sînt generate în Pasul 1, iar punctele $(0, R)$ şi $(0, -R)$ în Pasul 11. Aceasta permite organizarea corectă şi eficientă a principalelor faze de execuţie.

Primul test asupra variabilei s, prezent în algoritmul C1, nu mai este efectuat în algoritmul C2, deoarece la iniţializare $s = 1-R < 0$ pentru cercurile cu $R \geq 2$. Cercul de rază 1 nu poate fi generat de algoritmul C2 deoarece succesiunea de puncte (1, 0), (0, 1), (−1,0), (0, −1) nu îndeplineşte condiţia de conexiune discretă (secţiunea 8.1).

Fiecare fază a algoritmului conţine două bucle principale, corespunzătoare celor două tipuri de deplasări elementare efectuate la generarea punctelor într-un octant: o deplasare paralelă cu una din axe, respectiv o deplasare diagonală. În general, s-a urmărit ca operaţiile comune care trebuiesc efectuate pentru cele două deplasări elementare să apară o singură dată în algoritm, într-o parte comună a celor două bucle principale. De exemplu, pentru trasarea în octantul 1 există două bucle, alcătuite respectiv din paşii 2, 3 şi 3, 4; în Pasul 2 sînt incluse operaţiile referitoare numai la deplasarea în sus, în Pasul 4 operaţiile referitoare numai la deplasarea în diagonală, iar Pasul 3 conţine operaţiile care pot fi efectuate în comun: generarea efectivă a punctului şi adăugarea la s a cantităţii $v+1$, conform cu (3.36). Operaţia $v \leftarrow v+2$ trebuie să fie efectuată separat în fiecare buclă, datorită faptului că testul $u : v$ din Pasul 2 este necesar în situaţiile de excepţie prezentate în secţiunea 8.3.3.1.3.

Posibilitatea efectuării saltului din Pasul 2 la Pasul 9, în momentul cînd $u = v$, se bazează explicit pe relaţia $\Delta_n = -1$ existentă în acel moment, după cum a fost demonstrat în secţiunea 8.3.3.1.3. Prin urmare, $s = -1$, $s+1 = 0$ şi algoritmul alege continuarea corectă printr-o deplasare pe orizontală, fără generarea punctului de pe bisectoare.

8.3.5. Generarea cercurilor cu coordonatele centrului şi raza semiîntregi

Metoda de generare incrementală expusă în secţiunile precedente prezintă o oarecare lipsă de generalitate, datorită faptului că raza cercului este un număr întreg. În acest caz, diametrul cercului este un număr par de unităţi de rastru. În practică, poate să fie necesară o metodă de generare

eficientă a cercurilor cu diametre numere întregi oarecare, pentru care două puncte diametral opuse pot fi reprezentate exact în puncte din rastru. Din acest motiv, în această secţiune vom analiza pe scurt problema generării cercurilor centrate în puncte de coordonate semiîntregi şi cu raza de asemenea număr semiîntreg: $x_c = p + ½, y_c = q + ½, R = r + ½$, unde p, q, r sînt numere întregi pozitive.

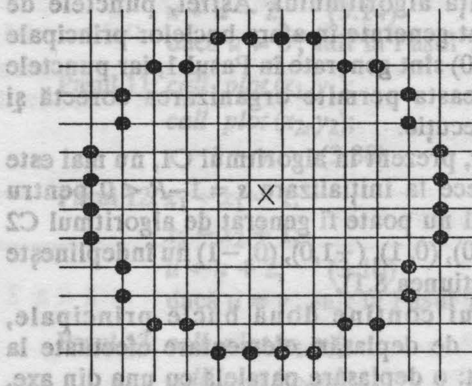


Figura 8–18.

Generarea prin puncte a unui cerc de rază 5 + ½ centrat într–un punct de coordonate semiîntregi.

Reprezentarea în rastru a unui astfel de cerc (fig. 8–18) prezintă proprietăţi de simetrie remarcabile, care sugerează posibilitatea dezvoltării unor algoritmi de generare eficienţi, la fel ca pentru cercurile de rază întreagă. Faptul că centrul nu corespunde unui pixel pe ecran nu deranjează, deoarece centrul nu intervine efectiv în procesul de generare; din contră, poziţia centrului reprezintă un avantaj, deoarece în acest caz punctele din rastru diametral opuse pe orizontală şi verticală sînt foarte aproape de curba ideală.

Evident, vom cerceta în ce măsură rezultatele teoretice obţinute în secţiunile 8.3.1, 8.3.2 şi 8.3.3 sînt aplicabile în această situaţie. Reluînd analiza din secţiunea 8.3.1 referitoare la primul octant, vom observa în primul rînd că toate coordonatele punctelor generate (x_n, y_n), pe care le vom considera tot relative la centrul cercului, sînt numere semiîntregi. Prin urmare, vom introduce notaţia:

$$x_n = p_n + ½ \,,\; y_n = q_n + ½ \,,\; p_n, q_n \text{ întregi.} \qquad (3.37)$$

Criteriul general (3.3) de efectuare a pasului incremental rămîne valabil, deoarece nu este influenţat de forma numerelor x_n, y_n. Să examinăm din nou mărimea Δ_n, definită în (3.5):

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 =$$

$$= (p_n+½)^2 + (p_n-½)^2 + 2(q_n+3/2)^2 - 2(r+½)^2 =$$

$$= 2p_n^2 + ½ + 2q_n^2 + 6q_n + 9/2 - 2r^2 - 2r - ½ =$$

$$= 2\,(p_n^2 + q_n^2 + 3q_n - r^2 - r + 2) + ½\,.$$

Aspectul important, care va fi util în continuare, este că numerele Δ_n sînt de forma $2k + ½$ (k întreg); vom spune că Δ_n sînt numere *pare semiîntregi*. Prin urmare, Δ_n nu poate fi nul și avem:

$$\begin{aligned} &\Delta_n > 0 \Rightarrow \Delta_n \geq ½ \\ &\Delta_n < 0 \Rightarrow \Delta_n \leq -3/2\,. \end{aligned} \tag{3.38}$$

La fel ca în secțiunea 8.3.1 se dovedește implicația $\delta_n \geq 0 \Rightarrow \Delta_n \geq 0$. În ceea ce privește implicația inversă, demonstrația prezentată în secțiunea 8.3.1 rămîne valabilă pînă la formula (3.7), pe care o reluăm aici:

$$8R^2\,[x_n^2 + (x_n-1)^2 + 2y_{n+1}^2 - 2R^2] = 8R^2\Delta_n < [x_n^2 - (x_n-1)^2]^2. \tag{3.39}$$

Nu am folosit pînă acum în demonstrație ipoteza $\Delta_n \geq 0$, care în acest caz implică $\Delta_n \geq ½$, conform cu (3.38). Atunci, din (3.39) rezultă:

$$[x_n^2 - (x_n-1)^2]^2 > 8R^2\,\Delta_n \geq 4R^2 \Rightarrow (2x_n-1)^2 > 4R^2 \Rightarrow$$

$$\Rightarrow 2x_n-1 > 2R$$

ceea ce este în contradicție cu condiția evidentă $x_n \leq R$. ■

Prin urmare, și în acest caz numerele δ_n și Δ_n au același semn, ceea ce înseamnă că criteriul (3.10) de efectuare a pasului incremental în octantul 1 rămîne valabil, dacă facem abstracție de faptul că numerele x_n, y_n, Δ_n nu sînt întregi.

În mod asemănător se demonstrează că formulele (3.14), (3.16), (3.18) de trasare incrementală a cercului în octanții 2, 3 și respectiv 4 rămîn valabile.

De asemenea, formulele (3.21), (3.24), (3.29) de trecere din octantul 1 în octantul 2 rămîn valabile, ca și formulele (3.31), (3.32), (3.33) de trecere din octantul 3 în octantul 4. Și în acest caz în procesul de generare incrementală pot apare situații de excepție, analizate în secțiunile 8.3.3.1.3 și 8.3.3.3.3, cînd algoritmul selectează un punct pe bisectoare după o deplasare orizontală sau verticală: de exemplu, pentru generarea cercului de rază 7 + ½. Formulele (3.29) și (3.33) asigură o funcționare corectă a algoritmului în aceste situații.

Singurele aspecte care trebuie examinate din nou, deoarece diferă față de rezultatele obținute în secțiunile anterioare, sînt valorile de inițializare a variabilelor x_n, y_n, Δ_n la începutul procesului de generare în octantul 1, și formulele de trecere din octantul 2 în octantul 3.

Evident, primul punct generat în octantul 1 are coordonatele $(x_0, y_0) = (R, ½)$ relative la centrul cercului, iar din formula (3.5) deducem:

$$\Delta_0 = x_0^2 + (x_0-1)^2 + 2(y_0+1)^2 - 2R^2 = R^2 + (R-1)^2 + 9/2 - 2R^2 =$$
$$= 1½ - 2R.$$

Prin urmare, iniţializarea variabilelor x_n, y_n, Δ_n se face după formulele:

$$x_0 = R \tag{3.40}$$
$$y_0 = ½$$
$$\Delta_0 = 1½ - 2R.$$

Formulele de intrare în octantul 3 se pot deduce uşor, dacă ţinem seama că ultimul punct generat în octantul 2 este $(x_n, y_n) = (½, R)$, iar primul punct care trebuie generat în octantul 3 este $(x_{n+1}, y_{n+1}) = (-½, R)$. În acest caz, din relaţiile (3.13) şi (3.15) rezultă:

$$\Delta_n = 2(x_n-1)^2 + (y_n+1)^2 + y_n^2 - 2R^2 =$$
$$= 2(½-1)^2 + (R+1)^2 + R^2 - 2R^2 = 3/2 + 2R$$
$$\Delta_{n+1} = 2(x_{n+1}-1)^2 + y_{n+1}^2 + (y_{n+1}-1)^2 - 2R^2 =$$
$$= 2(-½-1)^2 + R^2 + (R-1)^2 - 2R^2 =$$
$$= 1½ - 2R = 7 - \Delta_n.$$

Momentul traversării axei Oy se detectează cînd se selectează un punct cu abscisă negativă $x_{n+1} < 0$. Prin urmare, formulele de trecere din octantul 2 în octantul 3 sînt:

$$x_{n+1} < 0 \Rightarrow \Delta_{n+1} = 1½ - 2R = 7 - \Delta_n. \tag{3.41}$$

Pentru dezvoltarea unui algoritm concret de generare incrementală a cercurilor de rază semiîntreagă şi centrate în puncte de coordonate semiîntregi, utilizarea directă a variabilelor x_n, y_n, Δ_n nu este convenabilă, deoarece acestea nu sînt numere întregi. Vom introduce în acest scop 3 variabile de stare u, v şi s cu valori întregi, analoage celor folosite în algoritmul C2:

$$u = |2x_n| \tag{3.42}$$
$$v = 2y_n$$
$$s = \frac{\Delta_n - ½}{2}.$$

În definirea acestor variabile de stare am folosit explicit faptul că numerele x_n, y_n sînt semiîntregi, iar numerele Δ_n sînt pare semiîntregi.

Cu aceste variabile, se obţine un algoritm de generare foarte asemănător cu algoritmul C2. După cum rezultă din analiza anterioară,

singurele deosebiri apar în Pasul 1 care conține inițializarea variabilelor, conform cu (3.40), și în pașii 10 și 11 care testează condițiile de trecere din octantul 2 în octantul 3 și execută operațiile necesare, conform cu (3.41). În Pasul 11 trebuie adăugată operația $u \leftarrow -u$ $(=1)$ prin care se reface condiția corectă $u = |2x_n|$ după intrarea în octantul 3. De asemenea, în Pasul 20, traversarea axei Ox (= momentul terminării algoritmului) se detectează atunci cînd variabila v devine negativă $(v = -1)$; conform relațiilor de definiție (3.42), variabilele u și v sînt numere impare, deci nu iau niciodată valoarea 0.

În algoritmul de generare C3, descris în continuare, pașii care lipsesc sînt identici cu cei corespunzători din algoritmul C2.

Algoritmul C3

Generarea incrementală a unui cerc de centru $(x_c, y_c) = (p+½, q+½)$ de coordonate semiîntregi și de rază semiîntreagă $R = r+½ \geq 3+½$.

Pasul 1: Inițializări: $x_1 \leftarrow x_c + R, y_1 \leftarrow y_c + ½, x_2 \leftarrow x_c - R, y_2 \leftarrow y_c - ½$;
$u \leftarrow 2R, v \leftarrow 1, s \leftarrow 5/2 - R$; (3.40)
call plot (x_1, y_1);
call plot (x_2, y_2).
... ...
Pasul 10: $x_1 \leftarrow x_1 - 1$;
$x_2 \leftarrow x_2 + 1$;
$u \leftarrow u - 2$; (3.14)
dacă $u > 0$, salt la Pasul 7. (3.41)
Pasul 11: *call plot* (x_1, y_1);
call plot (x_2, y_2);
$u \leftarrow 1$;
$s \leftarrow 3 - s$. (3.41) (3.42)
... ...
Pasul 20: $y_1 \leftarrow y_1 - 1$;
$y_2 \leftarrow y_2 + 1$;
$v \leftarrow v - 2$; (3.18)
dacă $v > 0$, salt la Pasul 17.
Pasul 21: Stop.

8.3.6. Numărul de puncte generate

Cunoașterea numărului de puncte generate de algoritmul de trasare, care poate fi exprimat în funcție de raza cercului, permite deducerea pe

cale experimentală a vitezei de generare pe punct, pentru o implementare dată a algoritmului. Viteza de generare pe punct este un criteriu obiectiv de estimare a performanţelor algoritmului prezentat, în comparaţie cu alţi algoritmi de trasare rulaţi pe aceeaşi maşină.

În primul rînd, observăm că algoritmii C2 şi C3 generează acelaşi număr de puncte în fiecare octant; mai precis, punctele generate în fiecare octant se pot obţine din punctele generate în octantul 1, printr-o oglindire faţă de prima bisectoare şi/sau o rotaţie de 90°, 180° sau 270°. Se poate demonstra riguros că în octantul 2 se generează, în ordine inversă, simetricele faţă de prima bisectoare ale punctelor din octantul 1. Intuitiv, acest fapt se justifică astfel:

- curba ideală intersectează reţeaua de orizontale şi verticale a rastrului în puncte simetrice faţă de cele două bisectoare, cele două axe şi centrul cercului;
- metoda de generare prezentată asigură în fiecare octant cea mai bună aproximare discretă pentru curba ideală.

Prin urmare, pentru a calcula numărul de puncte $N(R)$ generate de algoritmul C2 pentru cercul de rază R, înmulţim cu 8 numărul de puncte generate în octantul 1.

Revenind la analizele din secţiunea 8.3.3, observăm că în situaţia din fig. 8–10 au fost generate punctele: (x_0,y_0) pe axa Ox, (x_1,y_1), (x_2,y_2), ...(x_n,y_n) în interiorul octantului 1, iar punctul (x_{n+1},y_{n+1}) este pe bisectoare. Prin urmare, pentru tot cercul vor fi generate $8n + 8$ puncte. Rămîne de găsit relaţia dintre n şi R în situaţia din fig. 8–10.

Deoarece în octantul 1 fiecare deplasare elementară are o componentă verticală, este clar că $y_n = n$ pentru orice punct generat. Rezultă că punctul A din fig. 8–10 are coordonatele:

$$(x_n,y_n) = (y_{n+2},y_n) = (n+2,n).$$

Aplicînd acestui punct relaţia $\Delta_n > 0$, premisă a deplasării diagonale de la A la B, obţinem:

$$\Delta_n = x_n^2 + (x_n-1)^2 + 2(y_n+1)^2 - 2R^2 = (n+2)^2 + 3(n+1)^2 - 2R^2 =$$

$$= 4n^2 + 10n + 7 - 2R^2 = (2n+2)(2n+3) + 1 - 2R^2 > 0. \quad (3.43)$$

Din a doua relaţie (3.12) aplicată în acelaşi punct A deducem:

$$\Delta_n - 4y_n - 2 = \Delta_n - 4n - 2 = 4n^2 + 6n + 5 - 2R^2 =$$

$$= (2n+1)(2n+2) + 3 - 2R^2 < 0. \quad (3.44)$$

Ţinînd seama că toate numerele sînt întregi, relaţiile (3.43) şi (3.44) pot fi scrise sub forma:

$$(2n+1)\,(2n+2) + 4 \le 2R^2 \le (2n+2)\,(2n+3)\,. \qquad (3.45)$$

Din relația (3.45) deducem succesiv:

$$(2n + 3/2)^2 + 6 - 9/4 \le 2R^2 \le (2n + 5/2)^2 + 6 - 25/4 \Rightarrow$$

$$\Rightarrow (2n + 3/2)^2 < 2R^2 < (2n + 5/2)^2 \Rightarrow 2n + 3/2 < R\sqrt{2} < 2n + 5/2 \Rightarrow$$

$$\Rightarrow 2n + 2 < R\sqrt{2} + 1/2 < 2n + 3\,.$$

Dubla inegalitate obținută este echivalentă cu egalitatea:

$$2n + 2 = \left[R\sqrt{2} + \frac{1}{2}\right]$$

unde am notat cu $[x]$ partea întreagă a numărului x. Prin urmare, algoritmul de trasare traversează prima bisectoare ca în fig. 8–10 dacă numărul $[R\sqrt{2} + 1/2]$, care este întregul cel mai apropiat de $R\sqrt{2}$, este par. În acest caz, ținînd seama că numărul total de puncte generate este $8n + 8$, rezultă:

$$N(R) = 4\left[R\sqrt{2} + \frac{1}{2}\right]. \qquad (3.46)$$

Altă posibilitate de traversare a bisectoarei este ilustrată în fig.8–11. În acest moment au fost generate punctele: (x_0, y_0) pe axa Ox, (x_1, y_1), (x_2, y_2), ...(x_n, y_n) în interiorul octantului 1. Punctul (x_{n+1}, y_{n+1}) este în interiorul octantului 2. Prin urmare, tot cercul va avea $8n + 4$ puncte.

Punctul A din fig. 8–11 are coordonatele:

$$(x_n, y_n) = (y_{n+1}, y_n) = (n+1, n)\,.$$

Aplicînd acestui punct aceleași condiții $\Delta_n > 0$ și $\Delta_n - 4y_n - 2 < 0$ justificate ca mai sus, deducem:

$$\Delta_n = x_n^2 + (x_n - 1)^2 + 2(y_n + 1)^2 - 2R^2 = 3(n+1)^2 + n^2 - 2R^2 =$$

$$= 4n^2 + 6n + 3 - 2R^2 = (2n+1)\,(2n+2) + 1 - 2R^2 > 0 \qquad (3.47)$$

$$\Delta_n - 4y_n - 2 = \Delta_n - 4n - 2 = 4n^2 + 2n + 1 - 2R^2 =$$

$$= 2n\,(2n+1) + 1 - 2R^2 < 0\,.$$

Deoarece toate numerele sînt întregi, relațiile (3.47) pot fi puse sub forma:

$$2n\,(2n+1) + 2 \le 2R^2 \le (2n+1)\,(2n+2) \qquad (3.48)$$

care este într–un fel complementară cu relația (3.45). Din (3.48) deducem succesiv:

$$(2n + 1/2)^2 + 2 - 1/4 \le 2R^2 \le (2n + 3/2)^2 + 2 - 9/4 \Rightarrow$$

$$\Rightarrow (2n + 1/2)^2 < 2R^2 < (2n + 3/2)^2 \Rightarrow 2n + 1/2 < R\sqrt{2} < 2n + 3/2 \Rightarrow$$

$$\Rightarrow 2n+1 < R\sqrt{2}+½ < 2n+2 \Rightarrow 2n+1 = \left[R\sqrt{2}+\frac{1}{2}\right].$$

Rezultă că algoritmul de generare traversează bisectoarele ca în fig. 8–11 dacă numărul $[R\sqrt{2} + ½]$ este impar. În acest caz, ţinînd seama că numărul total de puncte generate este $8n + 4$, rezultă pentru $N(R)$ aceeaşi expresie (3.46).

În sfîrşit, ultima posibilitate de traversare a primei bisectoare este cea din fig. 8–12, după o deplasare verticală. De fapt, tot $8n + 4$ puncte se generează şi în acest caz, deoarece punctele de pe bisectoare nu sînt efectiv generate, după cum s-a arătat în secţiunile 8.3.3.1.3 şi 8.3.3.3.3. Deosebirea este că în situaţia din fig. 8–12 avem $\Delta_n = -1$, în conformitate cu (3.27). Aceasta înseamnă că:

$$\Delta_n = 4n^2+6n+3-2R^2 = -1 \Rightarrow 2n^2+3n+2-R^2 = 0 \Rightarrow$$

$$\Rightarrow 2n+1 = \frac{\sqrt{8R^2-7}-1}{2}.$$

Se poate arăta că, dacă $(\sqrt{8R^2-7} - 1)/2$ este întreg, atunci:

$$\frac{\sqrt{8R^2-7}-1}{2} = \left[R\sqrt{2}-\frac{1}{2}\right].$$

Concluzia este că, dacă raza cercului este de forma specială precizată în secţiunea 8.3.3.1.3, soluţie a ecuaţiei în numere întregi (3.28), atunci numărul de puncte generate este:

$$N(R) = 4\left[R\sqrt{2}-\frac{1}{2}\right].$$

În toate celelalte cazuri, numărul de puncte generate de algoritmul C2 este:

$$N(R) = 4\left[R\sqrt{2}+\frac{1}{2}\right].$$

În mod asemănător se demonstrează că în cazul impar analizat în secţiunea 8.3.5, algoritmul C3 generează în total:

$$N'(R) = 4\left[\sqrt{2R^2-\frac{1}{2}}+\frac{1}{2}\right]$$

puncte, pentru orice valoare semiîntreagă a razei, cu excepţia valorilor de forma:

$$R_k = \frac{3}{4\sqrt{2}}\left\{(\sqrt{2}+1)^{4k+3}+(\sqrt{2}-1)^{4k+3}\right\},\ k = 0, 1, 2, 3, \ldots$$

Pentru aceste valori de excepţie ale razei, numărul de puncte generate de algoritmul C3 este:

$$N'(R) = 4\left[\sqrt{2R^2-\frac{1}{2}}-\frac{1}{2}\right] = 4\left[R\sqrt{2}-\frac{1}{2}\right].$$

8.4. Generarea incrementală a curbelor de gradul 2

În această secțiune vom căuta o metodă incrementală de trasare pentru curbele de gradul doi în forma cea mai generală, definită analitic printr-o ecuație de gradul 2 cu 2 necunoscute:

$$f(x,y) = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0 \qquad (4.1)$$

unde coeficienții $a_1, a_2, ..., a_6$ sînt în general numere reale. După cum este cunoscut, curba reprezentată de această ecuație este o (secțiune) *conică*.

8.4.1. Cîteva aspecte teoretice

Introducem notațiile:

$$\delta = a_2^2 - 4a_1a_3 \qquad (4.2)$$

$$\Delta = a_1a_5^2+a_3a_4^2-a_2a_4a_5+(a_2^2-4a_1a_3)\,a_6 = a_1a_5^2+a_3a_4^2-a_2a_4a_5+a_6\,\delta.$$

Deoarece Δ este o sumă de termeni de gradul 3 în funcție de coeficienții a_i, Δ își schimbă semnul dacă toți coeficienții își schimbă semnul. Pe de altă parte, este evident că o schimbare generală de semn a coeficienților nu afectează curba reprezentată de ecuația $f(x,y) = 0$. Prin urmare, fără restrîngerea generalității, putem să presupunem că este îndeplinită condiția:

$$\Delta \geq 0 \qquad (4.3)$$

care va fi utilă în cele ce urmează (eventual, este necesară o schimbare generală de semn a coeficienților a_i, $i = \overline{1,6}$ în faza de inițializare).

Vom spune că ecuația (4.1) reprezintă o *curbă degenerată* dacă funcția de gradul doi $f(x,y)$ se poate pune sub forma:

$$f(x,y) = (b_1x + b_2y + b_3)(c_1x + c_2y + c_3) \qquad (4.4)$$

unde coeficienții $b_1, b_2, b_3, c_1, c_2, c_3$ sînt în general numere complexe. Dacă toate numerele b_i, c_i sînt reale, conica (4.1) degenerează în două drepte reale, care pot fi concurente, paralele sau confundate, în funcție de anumite relații particulare între coeficienți. Dacă cel puțin unul din numerele b_i, c_i este imaginar, ecuația (4.1) nu reprezintă o curbă reală. În secțiunea 8.4.7 este prezentată o analiză detaliată a cazurilor de degenerare.

O condiție de degenerare echivalentă cu (4.4), dar mai ușor de testat, se obține cu ajutorul cantității Δ:

Teoremă. *Ecuația (4.1) reprezintă o conică degenerată dacă și numai dacă* $\Delta = 0$.

Demonstrație.

Dacă funcția $f(x,y)$ se poate pune sub forma (4.4), atunci egalînd coeficienții exprimărilor lui $f(x,y)$ din (4.1) și (4.4) obținem:

$$\begin{aligned} a_1 &= b_1c_1 \\ a_2 &= b_1c_2 + b_2c_1 \\ a_3 &= b_2c_2 \\ a_4 &= b_1c_3 + b_3c_1 \\ a_5 &= b_2c_3 + b_3c_2 \\ a_6 &= b_3c_3 . \end{aligned}$$

Pe baza acestor relații, se înlocuiesc numerele a_i cu b_i, c_i în expresia de definiție a lui Δ din (4.2) și rezultă ușor $\Delta = 0$.

Reciproc, să presupunem $\Delta = 0$. Să notăm:

$$\mu = 2\,(a_2 \pm \sqrt{a_2^2 - 4a_1a_3})$$

în care considerăm înaintea radicalului semnul efectiv al coeficientului a_2. Cantitatea μ poate fi reală sau complexă.

Dacă $\delta = a_2^2 - 4a_1a_3 \neq 0$, se arată ușor că $\mu \neq 0$ și prin calcul direct se verifică egalitatea:

$$f(x,y) = \frac{1}{\mu}\left[2a_1x + \frac{\mu}{2}y + \left(a_4 + \frac{a_2a_4 - 2a_1a_5}{\pm\sqrt{\delta}}\right)\right]\left[\frac{\mu}{2}x + 2a_3y + \left(a_5 + \frac{a_2a_5 - 2a_3a_4}{\pm\sqrt{\delta}}\right)\right].$$

Dacă $a_2^2 = 4a_1a_3$, în secțiunea 8.4.7 se demonstrează că $f(x,y)$ se poate scrie sub una din formele:

$$f(x,y) = \frac{1}{4a_1}\left(2a_1x + a_2y + a_4 + \sqrt{a_4^2 - 4a_1a_6}\right)\left(2a_1x + a_2y + a_4 - \sqrt{a_4^2 - 4a_1a_6}\right)$$

pentru $a_1 \neq 0$;

$$f(x,y) = \frac{1}{4a_3}\left(a_2x + 2a_3y + a_5 + \sqrt{a_5^2 - 4a_3a_6}\right)\left(a_2x + 2a_3y + a_5 - \sqrt{a_5^2 - 4a_3a_6}\right)$$

pentru $a_3 \neq 0$.

Prin urmare, în toate cazurile ecuația $f(x,y) = 0$ se poate descompune în două ecuații de gradul întîi, cu coeficienți în general complecși. ■

Dacă $\Delta \neq 0$, ecuația (4.1) reprezintă o curbă propriu-zisă, reală sau imaginară. Tipul conicei depinde de semnul discriminantului δ în modul următor:

$$\delta = a_2^2 - 4a_1a_3 \begin{cases} < 0 & - \text{ clipsă (sau cerc)} \\ = 0 & - \text{ parabolă} \\ > 0 & - \text{ hiperbolă}. \end{cases} \quad (4.5)$$

În cele ce urmează vom utiliza noţiunea de *curbură* $K(x,y)$ într–un punct (x,y) al curbei plane $f(x,y) = 0$. Dacă funcţia f are derivate parţiale de ordinul doi, curbura $K(x,y)$ este dată de relaţia:

$$K(x,y) = \frac{f''_{xx}(f'_y)^2 - 2f''_{xy}f'_xf'_y + f''_{yy}(f'_x)^2}{\left[(f'_x)^2 + (f'_y)^2\right]^{3/2}}. \quad (4.6)$$

Teoremă. *Pentru o curbă conică definită printr–o ecuaţie de forma (4.1), curbura $K(x,y)$ păstrează un semn constant în toate punctele curbei.*

Demonstraţie.

Calculînd derivatele parţiale ale funcţiei $f(x,y)$ şi ţinînd seama că (x,y) este un punct pe curba $f(x,y) = 0$, obţinem:

$$f''_{xx}(f'_y)^2 - 2f''_{xy}f'_xf'_y + f''_{yy}(f'_x)^2 =$$

$$= 2a_1(a_2x+2a_3y+a_5)^2 - 2a_2(2a_1x+a_2y+a_4)(a_2x+2a_3y+a_5) + $$
$$+ 2a_3(2a_1x+a_2y+a_4)^2 =$$

$$= 2(4a_1a_3-a_2^2)(a_1x^2+a_2xy+a_3y^2+a_4x+a_5y)+2a_1a_5^2-2a_2a_4a_5+2a_3a_4^2 =$$

$$= 2a_6(a_2^2-4a_1a_3)+2a_1a_5^2+2a_3a_4^2-2a_2a_4a_5 = 2\Delta \;;$$

$$K(x,y) = \frac{2\Delta}{\left[(f'_x)^2 + (f'_y)^2\right]^{3/2}} =$$

$$= \frac{2\Delta}{\left[(2a_1x+a_2y+a_4)^2 + (a_2x+2a_3y+a_5)^2\right]^{3/2}}. \quad (4.7)$$

Prin urmare, curbura $K(x,y)$ păstrează semnul lui Δ în toate punctele curbei, deci curba (4.1) nu are puncte de inflexiune. ■

Teoremă. *Dacă ecuaţia (4.1) reprezintă o curbă nedegenerată ($\Delta \neq 0$), atunci valoarea maximă a curburii K_{max} este:*

$$K_{max} = \frac{\left(a_1 + a_3 + \sqrt{a_1^2 + a_2^2 + a_3^2 - 2a_1a_3}\right)^{3/2}}{\sqrt{2\Delta}} = \quad (4.8)$$

$$= \frac{\left(a_1 + a_3 + \sqrt{a_1^2 + a_2^2 + a_3^2 - 2a_1a_3}\right)^{3/2}}{\sqrt{2}\left[a_1a_5^2 + a_3a_4^2 - a_2a_4a_5 + (a_2^2 - 4a_1a_3)\,a_6\right]^{1/2}}.$$

Demonstrație.

Din (4.7) rezultă că funcția $K(x,y)$ este maximă în punctele în care funcția ajutătoare $(f'_x)^2 + (f'_y)^2$ atinge valoarea minimă. Notînd $u = f'_x$, $v = f'_y$, rezultă că trebuie să calculăm extremele funcției:

$$g(u,v) = u^2 + v^2$$

cu condiția de legătură $f(x,y) = 0$. Prin calcul direct se dovedește că ecuația $f(x,y) = 0$ se poate pune mai convenabil sub forma:

$$a_1(f'_y)^2 - a_2 f'_x f'_y + a_3(f'_x)^2 = \Delta \Rightarrow \tag{4.9}$$

$$\Rightarrow h(u,v) = a_1 v^2 - a_2 uv + a_3 u^2 - \Delta = 0.$$

Aplicînd metoda multiplicatorilor lui Lagrange, definim funcția auxiliară:

$$\Phi(u,v,\lambda) = g(u,v) + \frac{1}{\lambda} h(u,v)$$

și determinăm punctele singulare ale acestei funcții.

$$\Phi'_u = 0 \Rightarrow 2\lambda u - a_2 v + 2a_3 u = 0 \Rightarrow 2(\lambda + a_3)u = a_2 v \tag{4.10}$$

$$\Phi'_v = 0 \Rightarrow 2\lambda v - a_2 u + 2a_1 v = 0 \Rightarrow 2(\lambda + a_1)v = a_2 u$$

$$\Phi'_\lambda = 0 \Rightarrow a_1 v^2 - a_2 uv + a_3 u^2 = \Delta.$$

Din primele două relații (4.10) rezultă ecuația care determină valoarea parametrului λ:

$$4(\lambda + a_1)(\lambda + a_3) = a_2^2 \tag{4.11}$$

care are rădăcinile:

$$\lambda_1 = \frac{-(a_1 + a_3) - \sqrt{a_1^2 + a_2^2 + a_3^2 - 2a_1 a_3}}{2} \tag{4.12}$$

$$\lambda_2 = \frac{-(a_1 + a_3) + \sqrt{a_1^2 + a_2^2 + a_3^2 - 2a_1 a_3}}{2}.$$

De asemenea, din primele două relații (4.10) obținem:

$$(\lambda + a_1)v^2 = (\lambda + a_3)u^2.$$

În acest caz, a 3–a relație (4.10) conduce la:

$$(\lambda + a_1)\Delta = a_1(\lambda + a_3)u^2 - \frac{a_2^2}{2}u^2 + a_3(\lambda + a_1)u^2 \Rightarrow$$

$$u^2 = \frac{2(\lambda + a_1)\Delta}{2(a_1 + a_3)\lambda + 4a_1 a_3 - a_2^2}, \quad v^2 = \frac{2(\lambda + a_3)\Delta}{2(a_1 + a_3)\lambda + 4a_1 a_3 - a_2^2}.$$

Prin urmare, valorile extreme ale funcției g sînt:

$$g(u,v) = u^2+v^2 = \frac{2(2\lambda+a_1+a_3)\Delta}{2(a_1+a_3)\lambda+4a_1a_3-a_2^2} = -\frac{\Delta}{\lambda},$$

ultima egalitate rezultînd din (4.11). Deoarece $\Delta > 0$, trebuie luate în considerare numai valorile negative ale lui λ. Distingem două cazuri:

- $\delta = a_2^2-4a_1a_3 < 0$ (elipse, cercuri). Rezultă în primul rînd că a_1, a_3 sînt nenule și de același semn, adică $a_1 > 0$, $a_3 > 0$ sau $a_1 < 0$, $a_3 < 0$. Dacă a_1 și a_3 sînt negative, egalitatea din (4.9) nu poate fi îndeplinită, deoarece cantitatea din membrul stîng păstrează semnul lui a_1 în orice punct (x,y) (dacă $a_1 < 0$, $a_3 < 0$, $\delta < 0$ și $\Delta > 0$, ecuația (4.1) nu definește o curbă reală). Prin urmare, $a_1 > 0$, $a_3 > 0$. Pe de altă parte, din $\delta < 0$ și din (4.12) rezultă $\lambda_1 \leq \lambda_2 < 0$. Există deci două valori extreme pentru funcția $g(u,v)$ (o valoare maximă și una minimă), și la fel pentru funcția de curbură $K(x,y)$. Din relația (4.7) deducem:

$$K_{max} = \frac{2\Delta}{(g_{min})^{3/2}} = \frac{2\Delta}{\left(-\frac{\Delta}{\lambda_1}\right)^{3/2}} = \frac{2(-\lambda_1)^{3/2}}{\sqrt{\Delta}} =$$

$$= \frac{\left(a_1+a_3+\sqrt{a_1^2+a_2^2+a_3^2-2a_1a_3}\right)^{3/2}}{\sqrt{2\Delta}}.$$

- $\delta = a_2^2-4a_1a_3 \geq 0$ (hiperbole, parabole). În acest caz, $|a_1+a_3| \leq \sqrt{a_1^2+a_2^2+a_3^2-2a_1a_3}$ și prin urmare $\lambda_1 < 0 \leq \lambda_2$. Curbura $K(x,y)$ are o singură valoare extremă (maximă), corespunzătoare lui λ_1. Pentru K_{max} se obține aceeași expresie (4.8). ■

În continuare demonstrăm un rezultat care va fi util în dezvoltarea unui algoritm de trasare eficient.

Teoremă. *Dacă ecuația (4.1) reprezintă o curbă reală nedegenerată ($\Delta > 0$), atunci funcția $f(x,y)$ ia valori negative în interiorul concavităților curbei $f(x,y) = 0$, și valori pozitive în exteriorul acestora.*

Demonstrație.

Fie (x_0,y_0) un punct pe curba $f(x,y) = 0$. Ecuația tangentei la curbă în punctul (x_0,y_0) este:

$$(x-x_0)f'_x(x_0,y_0) + (y-y_0)f'_y(x_0,y_0) = 0. \qquad (4.13)$$

Fie acum (x,y) un punct oarecare pe această tangentă (fig. 8–19). Pentru a evalua funcția f în punctul (x,y), folosim dezvoltarea în serie

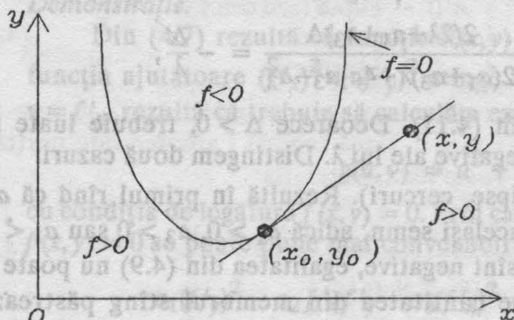


Figura 8–19.

Funcția f ia valori pozitive de–a lungul tangentei într–un punct (x_0,y_0) al curbei.

Taylor a lui f în jurul punctului (x_0,y_0), în care ne oprim la termenii de gradul doi, deoarece derivatele lui f de ordin superior sînt nule:

$$f(x,y) = f(x_0,y_0) + (x-x_0)f'_x(x_0,y_0) + (y-y_0)f'_y(x_0,y_0) +$$
$$+\tfrac{1}{2}(x-x_0)^2 f''_{xx}(x_0,y_0) + (x-x_0)(y-y_0)f''_{xy}(x_0,y_0) + \tfrac{1}{2}(y-y_0)^2 f''_{yy}(x_0,y_0).$$

Deoarece (x_0,y_0) este un punct pe curbă, iar (x,y) este un punct pe tangenta în (x_0,y_0), obținem:

$$f(x,y) = a_1(x-x_0)^2 + a_2(x-x_0)(y-y_0) + a_3(y-y_0)^2 . \tag{4.14}$$

În ipoteza $\Delta > 0$, din relația (4.9) rezultă că cel puțin una din derivatele parțiale f'_x, f'_y este nenulă în punctul (x_0,y_0). Presupunînd $f'_y(x_0,y_0) \neq 0$ (cazul $f'_x(x_0,y_0) \neq 0$ se tratează analog), din (4.13) și (4.14) deducem:

$$f(x,y) = \frac{(x-x_0)^2}{(f'_y)^2}\left[a_1(f'_y)^2 - a_2 f'_x f'_y + a_3(f'_x)^2\right],$$

toate derivatele fiind considerate în (x_0,y_0). Aplicînd în (x_0,y_0) relația (4.9), valabilă în orice punct al curbei, rezultă:

$$f(x,y) = \frac{\Delta(x-x_0)^2}{[f'_y(x_0,y_0)]^2} \geq 0 ;\ f(x,y) = 0 \Leftrightarrow (x,y) = (x_0,y_0) .$$

Rezultatul obținut dovedește că orice tangentă la o conică nu intersectează curba într–un alt punct, diferit de punctul de tangență; de–a lungul tangentei în (x_0,y_0), funcția f ia valori strict pozitive, cu excepția punctului (x_0,y_0) de pe curbă (fig. 8–19).

Reciproc, se poate arăta că dacă $f(x,y) > 0$ într–un punct (x,y) exterior curbei, atunci există un punct (x_0,y_0) pe curbă pentru care relațiile (4.14), și prin urmare (4.13), sînt îndeplinite, ceea ce înseamnă că dreapta care unește (x,y) și (x_0,y_0) este tangentă la curbă în (x_0,y_0).

Am dovedit deci că dintr–un punct (x, y) exterior curbei se poate duce o tangentă la curbă dacă și numai dacă $f(x, y) > 0$. Rezultă că funcția $f(x, y)$ ia valori strict pozitive în exteriorul concavităților curbei (4.1), și prin urmare valori strict negative în interiorul concavităților. ■

8.4.2. Ipoteze de lucru

Vom considera, dacă este necesar, aproximări cu numere raționale pentru coeficienții a_i, și apoi vom amplifica ecuația $f(x, y) = 0$ cu cel mai mic numitor comun al acestora. Prin urmare, menținînd problema într–un cadru suficient de general, putem să presupunem că în ecuația curbei $f(x, y) = 0$ coeficienții $a_1, a_2, ..., a_6$ sînt numere întregi, fără divizori comuni. În acest caz, funcția $f(x, y)$ ia valori întregi în punctele de coordonate (x, y) întregi, chiar dacă aceste puncte nu sînt situate pe curba $f(x, y) = 0$.

Metoda de trasare care va fi dezvoltată în secțiunile următoare utilizează implicit ipoteza de echidistanță orizontală și verticală a punctelor rastrului, discutată în secțiunea 8.1. Dacă dimensiunile celulei elementare de rastru sînt în raportul $\Delta x / \Delta y = p/q \neq 1$ (p, q întregi), pentru o trasare corectă se poate proceda în modul următor: se deduce ecuația curbei $g(x, y) = 0$ presupunînd că pasul rastrului în ambele direcții este egal cu pasul efectiv pe orizontală; apoi, înainte de aplicarea algoritmului de trasare, se înlocuiește ecuația $g(x, y) = 0$ cu:

$$f(x,y) = p^2 g\left(x, \frac{q}{p} y\right) = 0$$

care are tot coeficienți întregi și permite redarea corectă a proporțiilor.

La fel ca în secțiunile precedente, vom căuta o procedură incrementală de trasare a curbei, care să funcționeze prin deplasări elementare pe orizontală, verticală sau în diagonală între punctele succesiv generate. Vom numerota de la 0 la 7 direcțiile posibile de deplasare de la un punct fix P la unul din cele 8 puncte vecine din rastru $P_i, i = \overline{0,7}$ (fig. 8–20).

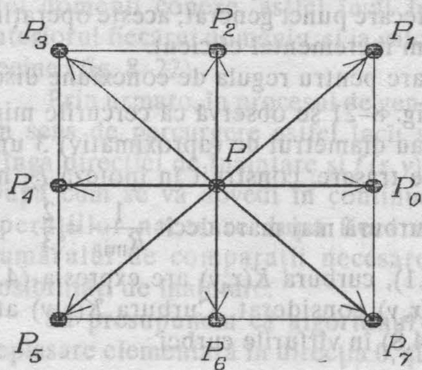


Figura 8–20.

Direcțiile posibile de deplasare elementară de la un punct P la un punct vecin din rastru, numerotate în sens trigonometric direct.

Pentru a putea dezvolta o procedură eficientă de generare incrementală a curbei generale (4.1), vom face două ipoteze rezonabile:

- **I1** Pentru două puncte (x_1,y_1) și (x_2,y_2) relativ apropiate de curba ideală, se poate decide care dintre ele este cel mai apropiat prin compararea cantităților $|f(x_1,y_1)|$ și $|f(x_2,y_2)|$. Prin urmare, vom considera că punctul cel mai apropiat de curbă este acela în care funcția $f(x,y)$ este minimă în valoare absolută.
- **I2** Dacă algoritmul de generare a efectuat o deplasare în direcția i între două puncte succesive (x_{n-1},y_{n-1}) și (x_n,y_n), atunci pentru următorul punct generat (x_{n+1},y_{n+1}) trebuie să se efectueze o deplasare elementară în una din direcțiile $i-1$, i sau $i+1$ ($i-1$ și $i+1$ sînt considerate modulo 8).

Ipoteza I_1 nu corespunde întotdeauna realității. De exemplu, să considerăm problema trasării cercului $f(x,y) = 32^2x^2 + 32^2y^2 - 113^2 = 0$. Curba ideală trece prin punctul $(113/32, 0)$, care este mai apropiat de $(4,0)$ decît de $(3,0)$. Pe de altă parte, algoritmul construit pe baza ipotezei I_1 va genera punctul $(3,0)$ în loc de $(4,0)$, deoarece $|f(3,0)| = 3553 < |f(4,0)| = 3615$. Acest exemplu dovedește că teorema demonstrată în secțiunile 8.3.1 și 8.3.5 pentru cazurile cînd raza cercului este un număr întreg, respectiv semiîntreg, nu este valabilă dacă raza este un număr fracționar oarecare.

Prin urmare, în ipoteza I_1, algoritmul nu va genera întotdeauna cea mai bună aproximare prin puncte din rastru pentru curba ideală. Totuși, în majoritatea situațiilor întîlnite în practică condiția din ipoteza I_1 este îndeplinită, iar în cazurile de excepție ca cel de mai sus, eroarea de plasare este minimă (cel mult un punct). Pe de altă parte, rezolvarea exactă a problemei plasării punctelor în toate cazurile implică în general rezolvarea unor ecuații de gradul doi pentru fiecare punct generat; aceste operații sînt prea complicate pentru un algoritm incremental eficient.

Ipoteza I_2 este o altă exprimare pentru regula de conexiune discretă prezentată în secțiunea 8.1. Din fig. 8–21 se observă că cercurile minime care îndeplinesc această condiție au diametrul de (aproximativ) 3 unități de rastru; rezultă că algoritmul de trasare, construit în ipoteza I_2, nu va aproxima corect curbele care au curbură mai mare decît $\frac{1}{R_{min}} = \frac{2}{3}$.

Pentru o curbă de forma (4.1), curbura $K(x,y)$ are expresia (4.7) și depinde în general de punctul (x,y) considerat. Curbura $K(x,y)$ atinge valoarea maximă dată de relația (4.8) în vîrfurile curbei.

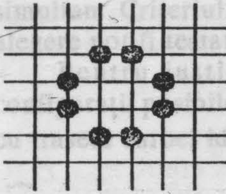


Figura 8–21.
În ipoteza I_2, cercurile minime care pot fi trasate au diametrul 3.

O posibilitate de a depăşi dificultăţile legate de curbură este să verificăm de la început condiţia $K_{max} \leq 2/3$, folosind expresia (4.8). O altă posibilitate este să atenuăm restricţia conţinută în ipoteza I_2, permiţînd algoritmului să efectueze schimbări de direcţie de 90° între două deplasări elementare succesive. Evident, această schimbare în ipoteza I_2 conduce la complicarea algoritmului şi la scăderea eficienţei în execuţie, deoarece sînt necesare mai multe comparaţii în bucla principală pentru selectarea direcţiei optime de înaintare (sînt 2 posibilităţi în plus faţă de cele 3 din ipoteza I_2).

Din punct de vedere practic, aceste precauţii nu sînt necesare, deoarece curbele de excentricitate foarte mare, care nu îndeplinesc condiţia $K_{max} \leq 2/3$, se întîlnesc rar în aplicaţiile reale. În plus, algoritmul de trasare proiectat în ipoteza I_2 evoluează acceptabil în aceste situaţii de excepţie.

Prin urmare, vom sacrifica exactitatea absolută în cazurile speciale şi degenerate, în favoarea eficienţei trasării pentru situaţiile practice obişnuite, şi vom proiecta algoritmul de trasare pe baza ipotezelor I_1 şi I_2 expuse mai sus.

8.4.3. Deducerea metodei de generare incrementală

În cazul general în care conica nu degenerează în două drepte concurente sau confundate, curba $f(x, y) = 0$ împarte planul în două sau trei domenii conexe, astfel încît funcţia f păstrează semn constant în interiorul fiecărui domeniu, şi ia valori de semne contrare în două domenii vecine (fig. 8–22).

Prin urmare, în procesul de generare a unui arc de curbă, putem alege un sens de parcurgere astfel încît să avem în permanenţă $f(x, y) < 0$ la stînga direcţiei de înaintare şi $f(x, y) > 0$ la dreapta direcţiei de înaintare. După cum se va dovedi în continuare, acest fapt permite simplificarea operaţiilor necesare după fiecare punct generat, prin minimizarea numărului de comparaţii necesare pentru alegerea uneia din cele 3 posibilităţi de înaintare.

Să presupunem că algoritmul a generat punctul $P = (x, y)$ după o deplasare elementară în direcţia 0, şi să analizăm ce operaţii sînt necesare pentru alegerea punctului următor. Conform ipotezei I_2, există trei puncte

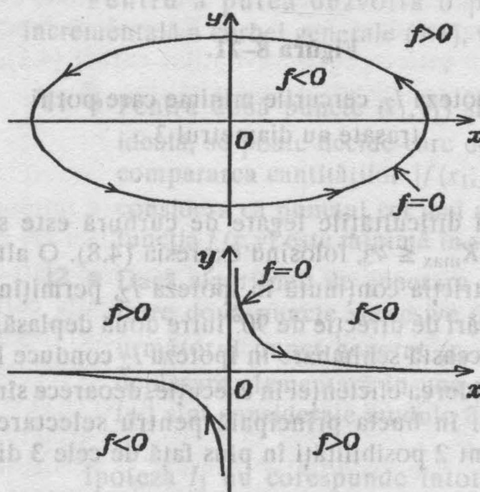


Figura 8–22.

Funcția f ia valori de semne contrare de o parte și de alta a curbei $f(x, y) = 0$.

candidat: $P_0 = (x+1, y)$, $P_1 = (x+1, y+1)$ și $P_7 = (x+1, y-1)$, cu notația din fig. 8–20. În baza ipotezei I_1, vom alege pe acela în care funcția f are cea mai mică valoare absolută.

Această alegere reclamă un șir destul de lung de comparații în cazul în care semnele cantităților care se compară nu sînt cunoscute. Dacă alegem sensul de parcurgere a curbei astfel încît $f(x,y) < 0$ (> 0) la stînga (respectiv dreapta) direcției de înaintare, atunci pentru alegerea unuia din cele 3 puncte candidat P_0, P_1, P_7 sînt necesare numai două comparații (numărul minim). Mai precis, vom considera următorul criteriu de alegere:

$$\begin{aligned} & f(P_7) + f(P_0) < 0 \Rightarrow \text{ se alege punctul } P_7; \\ & f(P_7) + f(P_0) \geq 0 \text{ și } f(P_0) + f(P_1) \leq 0 \Rightarrow \text{ se alege punctul } P_0; \\ & f(P_0) + f(P_1) > 0 \Rightarrow \text{ se alege punctul } P_1. \end{aligned} \tag{4.15}$$

Prin urmare, punctul P_7 este ales printr–o singură comparație, fără examinarea valorii funcției în P_1. Pentru a dovedi neambiguitatea acestui criteriu, trebuie să arătăm că prima și a treia condiție nu pot fi îndeplinite în același timp.

Să presupunem deci $f(P_7) + f(P_0) < 0$, $f(P_0) + f(P_1) > 0$. Din prima condiție rezultă $f(P_0) < 0$; într–adevăr, dacă $f(P_0) \geq 0$, atunci P_0 este la dreapta curbei (sau pe curbă); cu atît mai mult P_7 este la dreapta curbei $\Rightarrow f(P_7) \geq 0 \Rightarrow f(P_7) + f(P_0) \geq 0$.

Analog, din $f(P_0) + f(P_1) > 0$ deducem $f(P_0) > 0$; într–adevăr, $f(P_0) \leq 0 \Rightarrow P_0$ este la stînga curbei (sau pe curbă) $\Rightarrow P_1$ este la stînga curbei $\Rightarrow f(P_1) \leq 0 \Rightarrow f(P_0) + f(P_1) \leq 0$.

Contradicția obținută în privința semnului funcției f în punctul P_0 dovedește că prima și a treia condiție din (4.15) nu pot fi îndeplinite

simultan. Criteriul (4.15) este simetric, în sensul că cele 3 condiții de alegere pot fi testate în orice ordine. ■

Pentru justificarea criteriului (4.15), să considerăm cele 4 configurații posibile de poziționare relativă a punctelor P_0, P_1, P_7 în raport cu traseul curbei ideale $f(x,y) = 0$ (fig. 8–23):

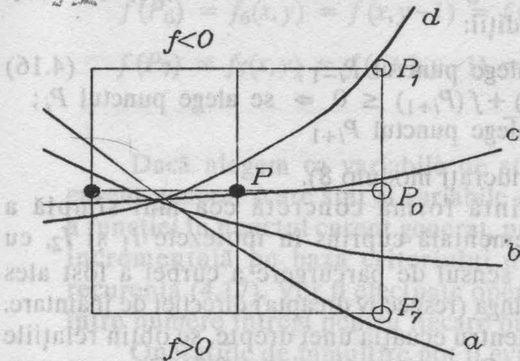


Figura 8–23.

Există 4 posibilități de trecere a curbei ideale printre punctele candidat P_0, P_1, P_7.

În cazul din fig. 8–23*a*, toate cele 3 puncte candidat sînt plasate la stînga traseului curbei ideale (punctul P_7 poate fi pe curbă); prin urmare, avem $f(P_0) < 0, f(P_1) < 0, f(P_7) \leq 0$ datorită modului de alegere a sensului de parcurgere a curbei. Criteriul (4.15) conduce, printr–o singură comparație, la alegerea punctului P_7, care este evident opțiunea optimă în ipoteza I_2.

Curba din fig. 8–23*b* trece printre punctele P_0 și P_7, prin urmare în acest caz avem $f(P_0) < 0, f(P_1) < 0, f(P_7) > 0$. Conform ipotezei I_1, trebuie să alegem punctul P_7 dacă $|f(P_7)| < |f(P_0)|$ și punctul P_0 dacă $|f(P_7)| \geq |f(P_0)|$. Semnele valorilor $f(P_0)$, $f(P_1)$, $f(P_7)$ fiind cunoscute, aceste inegalități sînt echivalente cu $f(P_7) < -f(P_0)$ și respectiv $f(P_7) \geq -f(P_0)$. Deoarece $f(P_0) + f(P_1) < 0$, rezultă că criteriul (4.15) asigură decizia corectă și în acest caz, prin una sau două comparații.

În situația din fig. 8–23*c*, curba ideală trece printre punctele P_0 și P_1 (P_0 poate fi pe curbă); în acest caz, avem $f(P_0) \geq 0$, $f(P_1) < 0$, $f(P_7) > 0$. Din ipoteza I_1 rezultă că trebuie să alegem punctul P_0 dacă $|f(P_0)| \leq |f(P_1)|$ și punctul P_1 dacă $|f(P_0)| > |f(P_1)|$. Aceste inegalități sînt echivalente cu $f(P_0) \leq -f(P_1)$, respectiv $f(P_0) > -f(P_1)$. Deoarece $f(P_7) + f(P_0) > 0$, rezultă că criteriul (4.15) conduce la alegerea corectă cu ajutorul a două comparații.

În sfîrșit, în situația din fig. 8–23*d*, toate cele 3 puncte candidat sînt situate la dreapta traseului curbei ideale (punctul P_1 poate fi pe curbă). În acest caz avem $f(P_0) > 0, f(P_1) \geq 0, f(P_7) > 0$. Prin urmare, criteriul (4.15) asigură decizia corectă de a alege punctul P_1, prin două comparații. ■

Criteriul (4.15) este valabil dacă punctul P a fost selectat în urma unei deplasări elementare în direcția 0; dar este evident că ideea de decizie este valabilă indiferent de direcția de înaintare. Prin urmare, criteriul (4.15) se poate generaliza în modul expus în continuare.

Dacă punctul P a fost generat în urma deplasării elementare în direcția i, atunci următorul punct generat este ales dintre P_{i-1}, P_i și P_{i+1}, prin testarea următoarelor condiții:

$$f(P_{i-1}) + f(P_i) < 0 \Rightarrow \text{se alege punctul } P_{i-1}; \quad (4.16)$$
$$f(P_{i-1}) + f(P_i) \geq 0 \text{ și } f(P_i) + f(P_{i+1}) \leq 0 \Rightarrow \text{se alege punctul } P_i;$$
$$f(P_i) + f(P_{i+1}) > 0 \Rightarrow \text{se alege punctul } P_{i+1}$$

(indicii $i-1$ și $i+1$ trebuie considerați modulo 8).

Criteriul (4.16) reprezintă forma concretă cea mai simplă a principiului de generare incrementală cuprins în ipotezele I_1 și I_2, cu presupunerea suplimentară că sensul de parcurgere a curbei a fost ales astfel încît $f(x,y) < 0$ (> 0) la stînga (respectiv dreapta) direcției de înaintare.

Aplicînd criteriul (4.16) pentru ecuația unei drepte, se obțin relațiile care stau la baza algoritmului lui Bresenham (secțiunea 8.2). De asemenea, particularizînd pentru cercurile cu raza număr întreg sau semiîntreg și analizînd separat fiecare octant, se dovedește ușor că criteriul general (4.16) este echivalent cu relațiile (3.10), (3.14), (3.16), (3.18) care guvernează trasarea incrementală a cercurilor. Prin urmare, metoda de trasare bazată pe criteriul de decizie (4.16) este o generalizare naturală a metodelor incrementale descrise în secțiunile anterioare pentru curbe mai simple.

8.4.4. Relații de recurență. Bucla principală a algoritmului de generare

Deoarece urmărim dezvoltarea unui algoritm incremental, presupunem cunoscută valoarea $f(x,y)$ a funcției într-un punct P și căutăm expresii simple pentru valorile funcției f în punctele imediat apropiate $P_i, i=\overline{0,7}$ (fig. 8-20). Dacă exprimăm toate valorile în funcție de coordonatele (x,y) ale punctului P și notăm $f_i(x,y) = f(P_i), i=\overline{0,7}$, atunci din expresia de definiție (4.1) rezultă:

$$f(P_0) = f_0(x,y) = f(x+1,y) = f(x,y) + (2a_1x+a_2y+a_4) + a_1 \quad (4.17)$$

$$f(P_1) = f_1(x,y) = f(x+1,y+1) = f(x,y) + (2a_1x+a_2y+a_4) + (a_2x+2a_3y+a_5) + a_1+a_2+a_3$$

$$f(P_2) = f_2(x,y) = f(x,y+1) = f(x,y) + (a_2x+2a_3y+a_5) + a_3$$

$$f(P_3) = f_3(x,y) = f(x-1,y+1) = f(x,y) - (2a_1x+a_2y+a_4) + (a_2x+2a_3y+a_5) + a_1-a_2+a_3$$

$$f(P_4) = f_4(x,y) = f(x-1,y) = f(x,y) - (2a_1x+a_2y+a_4) + a_1$$

$$f(P_5) = f_5(x,y) = f(x-1,y-1) = f(x,y) - (2a_1x+a_2y+a_4) - (a_2x+2a_3y+a_5) + a_1+a_2+a_3$$

$$f(P_6) = f_6(x,y) = f(x,y-1) = f(x,y) - (a_2x+2a_3y+a_5) + a_3$$

$$f(P_7) = f_7(x,y) = f(x+1,y-1) = f(x,y) + (2a_1x+a_2y+a_4) - (a_2x+2a_3y+a_5) + a_1-a_2+a_3 .$$

Dacă alegem ca variabile de stare coordonatele x,y ale punctului curent generat (care sînt și variabile de adresare fizică) și valoarea $f(x,y)$ a funcției în punctul curent generat, putem dezvolta un algoritm de trasare incrementală pe baza criteriului de decizie (4.16) și a relațiilor de recurență (4.17). Vor fi efectuate operații de adunare, scădere și înmulțire între numere întregi pentru fiecare punct generat.

Operațiile de înmulțire pot fi evitate în cazul unei alegeri mai atente a variabilelor de stare. Observăm că relațiile de recurență (4.17) pot fi exprimate mai simplu în funcție de derivatele parțiale f'_x și f'_y ale lui f în punctul $P = (x,y)$, pe care le notăm fx, respectiv fy:

$$fx(x,y) = f'_x(x,y) = 2a_1x + a_2y + a_4 \qquad (4.18)$$
$$fy(x,y) = f'_y(x,y) = a_2x + 2a_3y + a_5 .$$

Funcțiile fx, fy fiind liniare, actualizările lor incrementale se exprimă numai prin operații de adunare și scădere, în cazul unei deplasări elementare între două puncte vecine:

$$fx(P_0) = fx_0(x,y) = fx(x+1,y) = fx(x,y) + 2a_1 \qquad (4.19)$$
$$fy(P_0) = fy_0(x,y) = fy(x+1,y) = fy(x,y) + a_2$$
$$fx(P_1) = fx_1(x,y) = fx(x+1,y+1) = fx(x,y) + 2a_1 + a_2$$
$$fy(P_1) = fy_1(x,y) = fy(x+1,y+1) = fy(x,y) + a_2 + 2a_3$$

etc.

Este evident acum că trebuie să alegem ca variabile de stare valorile funcțiilor f, fx și fy în punctul curent generat (x,y). Din (4.17) și (4.19) rezultă formulele de modificare incrementală a acestor variabile de stare în cazul unei deplasări elementare în direcția $i, i = \overline{0,7}$. Sînt necesare numai operații de adunare și scădere între numere întregi:

$$\begin{cases} f_0 = f + fx + a_1 \\ fx_0 = fx + 2a_1 \\ fy_0 = fy + a_2 \end{cases} \qquad (4.20)$$

$$\begin{cases} f_1 = f + fx + fy + a_1 + a_2 + a_3 \\ fx_1 = fx + 2a_1 + a_2 \\ fy_1 = fy + a_2 + 2a_3 \end{cases}$$

$$\begin{cases} f_2 = f + fy + a_3 \\ fx_2 = fx + a_2 \\ fy_2 = fy + 2a_3 \end{cases}$$

$$\begin{cases} f_3 = f - fx + fy + a_1 - a_2 + a_3 \\ fx_3 = fx - 2a_1 + a_2 \\ fy_3 = fy - a_2 + 2a_3 \end{cases}$$

$$\begin{cases} f_4 = f - fx + a_1 \\ fx_4 = fx - 2a_1 \\ fy_4 = fy - a_2 \end{cases}$$

$$\begin{cases} f_5 = f - fx - fy + a_1 + a_2 + a_3 \\ fx_5 = fx - 2a_1 - a_2 \\ fy_5 = fy - a_2 - 2a_3 \end{cases}$$

$$\begin{cases} f_6 = f - fy + a_3 \\ fx_6 = fx - a_2 \\ fy_6 = fy - 2a_3 \end{cases}$$

$$\begin{cases} f_7 = f + fx - fy + a_1 - a_2 + a_3 \\ fx_7 = fx + 2a_1 - a_2 \\ fy_7 = fy + a_2 - 2a_3 \end{cases}$$

(pentru simplitate, nu au mai fost figurate argumentele x,y pentru funcțiile $f, fx, fy, f_i, fx_i, fy_i$).

Pe baza formulelor (4.20), se poate dezvolta ușor algoritmul de trasare incrementală în partea lui cea mai importantă, care este bucla principală de generare a punctelor. Pentru obținerea eficienței maxime, sînt necesare secvențe separate pentru cele 8 deplasări elementare posibile între două puncte generate consecutiv.

Deoarece curba reprezentată de ecuația (4.1) poate fi infinită (parabolă sau hiperbolă), dar spațiul de afișare (ecranul display–ului) este finit, problema încadrării în fereastra de vizualizare („clipping") nu mai poate fi neglijată. Din acest motiv, în secvențele care implementează cele 8 deplasări elementare posibile trebuie incluse teste de depășire a limitelor ferestrei de vizualizare, corespunzătoare deplasărilor respective.

Pentru definirea ferestrei de vizualizare, vom introduce 4 constante întregi *xmin*, *ymin*, *xmax*, *ymax*, precizate astfel: (*xmin*, *ymin*) este adresa punctului limită stînga–jos, iar (*xmax*, *ymax*) este adresa punctului limită dreapta–sus, reprezentabile în fereastra de vizualizare. În mod normal, $xmin = 0, ymin = 0, xmax = xres-1, ymax = yres-1$, unde *xres*, *yres* reprezintă rezoluția ecranului în pixeli, pe orizontală, respectiv verticală.

Numerele *xmin*, *ymin*, *xmax*, *ymax* reprezintă limitele domeniului de variație pentru variabilele de adresare fizică x, y. De exemplu, pentru un display cu rezoluția de 1024×768 puncte, vom avea:

$$0 = xmin \leq x \leq xmax = 1023$$
$$0 = ymin \leq y \leq ymax = 767\,.$$

După aceste precizări, prezentăm secvențele care implementează deplasările elementare în cele 8 direcții, precum și modificările corespunzătoare ale variabilelor de stare f, fx, fy; aceste secvențe constituie împreună nucleul algoritmului de trasare incrementală a unei curbe plane de gradul doi.

Secvența D0

Deplasare elementară în direcția 0 (la dreapta).

Pasul 1: *call plot*1 (x,y);
dacă $x = xmax$, return.

Pasul 2: $x \leftarrow x + 1$;
$fx \leftarrow fx + 2a_1$;
$fy \leftarrow fy + a_2$.

Pasul 3: $f_7 \leftarrow f + fx - fy + a_1 - a_2 + a_3$;
$f_0 \leftarrow f + fx + a_1$;
dacă $f_7 + f_0 < 0$, atunci $f \leftarrow f_7$ și salt la Secvența D7.

Pasul 4: $f_1 \leftarrow f + fx + fy + a_1 + a_2 + a_3$;
dacă $f_0 + f_1 \leq 0$, atunci $f \leftarrow f_0$ și salt la Secvența D0.

Pasul 5: $f \leftarrow f_1$ și salt la Secvența D1.

În primul pas se generează efectiv un punct în mediul de afișare și se verifică condiția de încadrare în fereastra de vizualizare. Subrutina *plot*1 realizează o operație în plus față de subrutina *plot*, și va fi precizată ulterior (secțiunea 8.4.6). În Pasul 2 se actualizează variabilele de adresare fizică și variabilele de stare fx și fy, corespunzător deplasării elementare în direcția 0. În acest moment variabila f este deja actualizată (la ieșirea din secvența de deplasare anterioară). Pașii 3, 4 și 5 implementează criteriul de decizie (4.15), pe baza relațiilor de recurență (4.20) între variabilele de stare. Înainte de trecerea la următoarea secvență de deplasare elementară, se actualizează variabila de stare f.

Pentru celelalte direcții, secvențele de generare–deplasare se dezvoltă în mod analog, după cum urmează:

Secvența D1

Deplasare elementară în direcția 1 (dreapta–sus).

Pasul 1: *call plot*1 (x, y);
dacă $x = xmax$, return;
dacă $y = ymax$, return.

Pasul 2: $x \leftarrow x + 1, y \leftarrow y + 1$;
$fx \leftarrow fx + 2a_1 + a_2$;
$fy \leftarrow fy + a_2 + 2a_3$.

Pasul 3: $f_0 \leftarrow f + fx + a_1$;
$f_1 \leftarrow f + fx + fy + a_1 + a_2 + a_3$;
dacă $f_0 + f_1 < 0$, atunci $f \leftarrow f_0$ și salt la Secvența D0.

Pasul 4: $f_2 \leftarrow f + fy + a_3$;
dacă $f_1 + f_2 \leq 0$, atunci $f \leftarrow f_1$ și salt la Secvența D1.

Pasul 5: $f \leftarrow f_2$ și salt la Secvența D2.

Secvența D2

Deplasare elementară în direcția 2 (în sus).

Pasul 1: *call plot*1 (x, y);
dacă $y = ymax$, return.

Pasul 2: $y \leftarrow y + 1$;
$fx \leftarrow fx + a_2$;
$fy \leftarrow fy + 2a_3$.

Pasul 3: $f_1 \leftarrow f + fx + fy + a_1 + a_2 + a_3$;
$f_2 \leftarrow f + fy + a_3$;
dacă $f_1 + f_2 < 0$, atunci $f \leftarrow f_1$ și salt la Secvența D1.

Pasul 4: $f_3 \leftarrow f - fx + fy + a_1 - a_2 + a_3$;
dacă $f_2 + f_3 \leq 0$, atunci $f \leftarrow f_2$ și salt la Secvența D2.

Pasul 5: $f \leftarrow f_3$ și salt la Secvența D3.

Secvența D3

Deplasare elementară în direcția 3 (stînga–sus).

Pasul 1: *call plot*1 (x, y);

dacă $x = xmin$ sau $y = ymax$, return.

Pasul 2: $x \leftarrow x - 1, y \leftarrow y + 1$;
$fx \leftarrow fx - 2a_1 + a_2$;
$fy \leftarrow fy - a_2 + 2a_3$.

Pasul 3: $f_2 \leftarrow f + fy + a_3$;
$f_3 \leftarrow f - fx + fy + a_1 - a_2 + a_3$;
dacă $f_2 + f_3 < 0$, atunci $f \leftarrow f_2$ și salt la Secvența D2.

Pasul 4: $f_4 \leftarrow f - fx + a_1$;
dacă $f_3 + f_4 \leq 0$, atunci $f \leftarrow f_3$ și salt la Secvența D3.

Pasul 5: $f \leftarrow f_4$ și salt la Secvența D4.

Secvența D4

Deplasare elementară în direcția 4 (la stînga).

Pasul 1: *call plot*1 (x,y);
dacă $x = xmin$, return.

Pasul 2: $x \leftarrow x - 1$;
$fx \leftarrow fx - 2a_1$;
$fy \leftarrow fy - a_2$.

Pasul 3: $f_3 \leftarrow f - fx + fy + a_1 - a_2 + a_3$;
$f_4 \leftarrow f - fx + a_1$;
dacă $f_3 + f_4 < 0$, atunci $f \leftarrow f_3$ și salt la Secvența D3.

Pasul 4: $f_5 \leftarrow f - fx - fy + a_1 + a_2 + a_3$;
dacă $f_4 + f_5 \leq 0$, atunci $f \leftarrow f_4$ și salt la Secvența D4.

Pasul 5: $f \leftarrow f_5$ și salt la Secvența D5.

Secvența D5

Deplasare elementară în direcția 5 (stînga–jos).

Pasul 1: *call plot*1 (x,y);
dacă $x = xmin$, return;
dacă $y = ymin$, return.

Pasul 2: $x \leftarrow x - 1, y \leftarrow y - 1$;
$fx \leftarrow fx - 2a_1 - a_2$;
$fy \leftarrow fy - a_2 - 2a_3$.

Pasul 3: $f_4 \leftarrow f - fx + a_1$;

$f_5 \leftarrow f - fx - fy + a_1 + a_2 + a_3$;
dacă $f_4 + f_5 < 0$, atunci $f \leftarrow f_4$ și salt la Secvența D4.

Pasul 4: $f_6 \leftarrow f - fy + a_3$;
dacă $f_5 + f_6 \leq 0$, atunci $f \leftarrow f_5$ și salt la Secvența D5.

Pasul 5: $f \leftarrow f_6$ și salt la Secvența D6.

Secvența D6

Deplasare elementară în direcția 6 (în jos).

Pasul 1: *call plot*1 (x, y);
dacă $y = ymin$, return.

Pasul 2: $y \leftarrow y - 1$;
$fx \leftarrow fx - a_2$;
$fy \leftarrow fy - 2a_3$.

Pasul 3: $f_5 \leftarrow f - fx - fy + a_1 + a_2 + a_3$;
$f_6 \leftarrow f - fy + a_3$;
dacă $f_5 + f_6 < 0$, atunci $f \leftarrow f_5$ și salt la Secvența D5.

Pasul 4: $f_7 \leftarrow f + fx - fy + a_1 - a_2 + a_3$;
dacă $f_6 + f_7 \leq 0$, atunci $f \leftarrow f_6$ și salt la Secvența D6.

Pasul 5: $f \leftarrow f_7$ și salt la Secvența D7.

Secvența D7

Deplasare elementară în direcția 7 (dreapta–jos).

Pasul 1: *call plot*1 (x, y);
dacă $x = xmax$ sau $y = ymin$, return.

Pasul 2: $x \leftarrow x + 1, y \leftarrow y - 1$;
$fx \leftarrow fx + 2a_1 - a_2$;
$fy \leftarrow fy + a_2 - 2a_3$.

Pasul 3: $f_6 \leftarrow f - fy + a_3$;
$f_7 \leftarrow f + fx - fy + a_1 - a_2 + a_3$;
dacă $f_6 + f_7 < 0$, atunci $f \leftarrow f_6$ și salt la Secvența D6.

Pasul 4: $f_0 \leftarrow f + fx + a_1$;
dacă $f_7 + f_0 \leq 0$, atunci $f \leftarrow f_7$ și salt la Secvența D7.

Pasul 5: $f \leftarrow f_0$ și salt la Secvența D0.

Rutinele prezentate mai sus utilizează, în afară de variabilele de stare f, fx, fy şi variabilele de adresare fizică x, y, încă 8 variabile auxiliare temporare $f_i, i = \overline{0,7}$. Acestea pun în evidenţă modul de folosire a formulelor de deplasare incrementală (4.20) pentru actualizarea variabilelor de stare. De fapt, este suficientă o singură variabilă auxiliară, comună pentru toate cele 8 secvenţe, şi această posibilitate trebuie luată în considerare în cazul unei implementări concrete. De exemplu, secvenţa care realizează deplasarea elementară în direcţia 0 poate fi rescrisă într–un mod echivalent, cu ajutorul variabilei auxiliare t, în modul următor:

Secvenţa D0

Deplasare elementară în direcţia 0 (la dreapta).

Pasul 1: *call plot*1 (x,y);
dacă $x = xmax$, return.

Pasul 2: $x \leftarrow x + 1$;
$fx \leftarrow fx + 2a_1$;
$fy \leftarrow fy + a_2$.

Pasul 3: $t \leftarrow f + fx + a_1$;
$f \leftarrow t - fy - a_2 + a_3$;
dacă $f+t < 0$, atunci salt la Secvenţa D7.

Pasul 4: $f \leftarrow t + fy + a_2 + a_3$;
dacă $f+t > 0$, atunci salt la Secvenţa D1.

Pasul 5: $f \leftarrow t$ şi salt la Secvenţa D0.

În plus, secvenţa precedentă ilustrează posibilităţile de reducere a numărului operaţiilor aritmetice şi de atribuire necesare pentru implementarea deplasărilor incrementale.

8.4.5. Trasarea unui arc de curbă

În această secţiune prezentăm procedurile de trasare a unui arc de curbă de ecuaţie (4.1), pornind de la un punct (x, y) de intersecţie al curbei cu una din laturile ferestrei de vizualizare. Se presupune că sensul de parcurgere a arcului de curbă a fost ales astfel încît $f < 0$ ($f > 0$) la stînga (respectiv dreapta) direcţiei de înaintare.

Procedurile de trasare a unui arc de curbă se bazează pe secvenţele de generare incrementală D0, D1, ... D7 dezvoltate în secţiunea 8.4.4. La început, fiecare procedură de trasare trebuie să iniţializeze variabilele de stare f, fx, fy şi să aleagă direcţia primei deplasări elementare. Deplasările

ulterioare sînt efectuate în cadrul secvenţelor D0, D1, ... D7, pînă cînd se ajunge la celălalt capăt al arcului de curbă, adică pînă cînd se întîlneşte din nou o margine a ferestrei de vizualizare.

În continuare descriem procedurile *arc_up* şi *arc_up_rev*, care trasează un arc de curbă pornind de la un punct situat la marginea de jos ($y = ymin$) a ferestrei de vizualizare. În acest caz, prima deplasare elementară poate fi în una din direcţiile 0, 1, 2, 3 sau 4. Pentru alegerea direcţiei optime, poate fi aplicată ideea cuprinsă în criteriul (4.16) de efectuare a pasului incremental. Mai precis, se calculează pe rînd sumele $f(P_i) + f(P_{i+1})$, pentru $i = 0, 1, 2, 3$ şi se alege deplasarea în direcţia j, unde j este primul i pentru care $f(P_i) + f(P_{i+1}) < 0$. În acest caz sînt necesare (maximum) 4 comparaţii, deoarece există 5 posibilităţi de alegere. Acest principiu de decizie se bazează în mod esenţial pe ipoteza că $f < 0$ (> 0) la stînga (respectiv dreapta) direcţiei de înaintare (fig. 8–24).

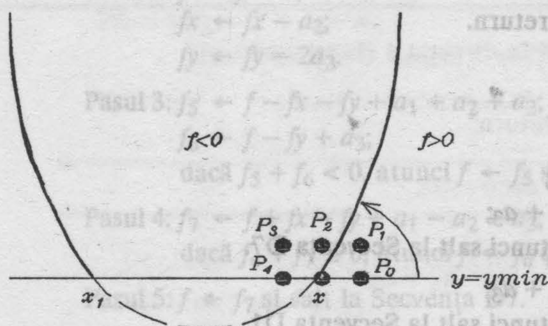


Figura 8–24. Alegerea primei deplasări elementare prin inspectarea direcţiilor posibile în sens trigonometric direct.

Criteriul de alegere de mai sus reprezintă de fapt generalizarea criteriului (4.16) pentru 5 puncte candidat. La fel ca în secţiunea 8.4.3, se poate obţine uşor o justificare riguroasă analizînd toate posibilităţile de poziţionare relativă a punctelor candidat P_0, P_1, P_2, P_3, P_4 în raport cu traseul curbei ideale. Procedura de trasare a unui arc de curbă descrisă în continuare se bazează pe metoda prezentată mai sus de alegere a direcţiei iniţiale.

Procedura arc_up

Trasarea unui arc de curbă pornind de la un punct (x,y) situat la marginea de jos a ferestrei de vizualizare ($y = ymin$).

Pasul 1: $f \leftarrow a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6$;
$fx \leftarrow 2a_1x + a_2y + a_4$.

Pasul 2: $f_0 \leftarrow f + fx + a_1$;
dacă $f + f_0 \geq 0$, atunci salt la Pasul 3;

$x \leftarrow x + 1$;
$f \leftarrow f_0$;
$fx \leftarrow fx + 2a_1$.

Pasul 3: dacă $x < xmin$, return;
dacă $x > xmax$, return.

Pasul 4: $fy \leftarrow a_2x + 2a_3y + a_5$;
drawn ← *true*.

Pasul 5: $f_0 \leftarrow f + fx + a_1$;
$f_1 \leftarrow f + fx + fy + a_1 + a_2 + a_3$;
dacă $f_0 + f_1 < 0$, atunci $f \leftarrow f_0$ și salt la Secvența D0.

Pasul 6: $f_2 \leftarrow f + fy + a_3$;
dacă $f_1 + f_2 < 0$, atunci $f \leftarrow f_1$ și salt la Secvența D1.

Pasul 7: $f_3 \leftarrow f - fx + fy + a_1 - a_2 + a_3$;
dacă $f_2 + f_3 \leq 0$, atunci $f \leftarrow f_2$ și salt la Secvența D2.

Pasul 8: $f_4 \leftarrow f - fx + a_1$;
dacă $f_3 + f_4 \leq 0$, atunci $f \leftarrow f_3$ și salt la Secvența D3.

Pasul 9: $f \leftarrow f_4$ și salt la Secvența D4.

Procedura are ca date de intrare coordonatele x,y ale punctului inițial, care se calculează în programul principal (secțiunea 8.4.6) prin intersectarea curbei cu dreapta $y = ymin$. În Pasul 1 se inițializează variabilele de stare $f = f(x,y)$ și $fx = f'_x(x,y)$. În Pasul 2 se efectuează o corecție a valorii abscisei x, dat fiind că valoarea inițială este calculată în programul principal cu ajutorul funcției parte întreagă, deci cu aproximație prin lipsă. Variabila x este incrementată dacă punctul $(x+1,ymin)$ este mai aproape de curbă decît $(x,ymin)$, ceea ce se detectează prin condiția $f(x,ymin) + f(x+1,ymin) < 0$. În Pasul 3 se verifică condițiile de încadrare în fereastra de vizualizare pe orizontală, care nu sînt verificate în programul principal. În Pasul 4 se inițializează variabila de stare $fy = f'_y(x,y)$ și se memorează în variabila logică globală *drawn* faptul că un arc de curbă este efectiv generat. Rolul acestei variabile globale va fi precizat în secțiunea 8.4.6. Pașii 5, 6, 7, 8, 9 implementează criteriul de alegere a direcției inițiale de deplasare expus mai înainte.

Procedura *arc_up* funcționează corect în cazul în care $x_1 \leq x$, unde x_1 este abscisa celui de-al doilea punct de intersecție al curbei (4.1) cu orizontala $y = ymin$ (fig. 8–24). Chiar dacă cele două puncte de intersecție (dacă acestea există) sînt foarte apropiate sau coincid din cauza aproximărilor la numere întregi, procedura *arc_up* selectează direcția inițială corectă.

Pe de altă parte, dacă al doilea punct de intersecție este la dreapta

punctului $(x, ymin)$, ca în fig. 8–25, procedura *arc_up* este inoperantă în cazul cînd cele două puncte de intersecţie sînt foarte apropiate. În această situaţie este necesară examinarea celor 5 posibilităţi în ordine inversă, adică în sens trigonometric invers. Prin urmare, procedeul corect de alegere a direcţiei iniţiale constă în a calcula pe rînd sumele $f(P_i) + f(P_{i-1})$, pentru $i = 4, 3, 2, 1$, şi a selecta direcţia iniţială j, unde j este primul (cel mai mare) i pentru care $f(P_i) + f(P_{i-1}) > 0$; acest criteriu de decizie se concretizează în procedura *arc_up_rev* descrisă în continuare.

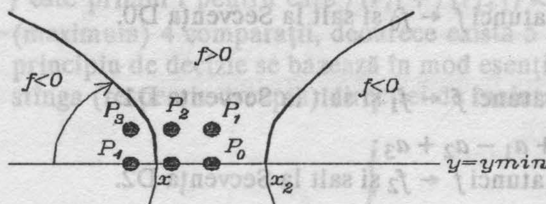


Figura 8–25.

Alegerea primei deplasări elementare prin examinarea direcţiilor posibile în sens trigonometric invers.

Procedura arc_up_rev

Trasarea unui arc de curbă pornind de la un punct (x, y) situat la marginea de jos a ferestrei de vizualizare $(y = ymin)$.

Pasul 1: $f \leftarrow a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6$;
$fx \leftarrow 2a_1x + a_2y + a_4$.

Pasul 2: $f_0 \leftarrow f + fx + a_1$;
dacă $f + f_0 \geq 0$, atunci salt la Pasul 3;
$x \leftarrow x + 1$;
$f \leftarrow f_0$;
$fx \leftarrow fx + 2a_1$.

Pasul 3: dacă $x < xmin$, return;
dacă $x > xmax$, return.

Pasul 4: $fy \leftarrow a_2x + 2a_3y + a_5$;
drawn ← *.true.*

Pasul 5: $f_4 \leftarrow f - fx + a_1$;
$f_3 \leftarrow f - fx + fy + a_1 - a_2 + a_3$;
dacă $f_4 + f_3 > 0$, atunci $f \leftarrow f_4$ şi salt la Secvenţa D4.

Pasul 6: $f_2 \leftarrow f + fy + a_3$;
dacă $f_3 + f_2 > 0$, atunci $f \leftarrow f_3$ şi salt la Secvenţa D3.

Pasul 7: $f_1 \leftarrow f + fx + fy + a_1 + a_2 + a_3$;
dacă $f_2 + f_1 \geq 0$, atunci $f \leftarrow f_2$ și salt la Secvența D2.

Pasul 8: $f_0 \leftarrow f + fx + a_1$;
dacă $f_1 + f_0 \geq 0$, atunci $f \leftarrow f_1$ și salt la Secvența D1.

Pasul 9: $f \leftarrow f_0$ și salt la Secvența D0.

Procedurile *arc_up* și *arc_up_rev* se referă la un arc de curbă care pornește de la marginea de jos a ferestrei. Pentru fiecare din celelalte 3 margini sînt necesare cîte două proceduri: *arc_down* și *arc_down_rev* pentru marginea de sus, *arc_right* și *arc_right_rev* pentru marginea din stînga, *arc_left* și *arc_left_rev* pentru marginea din dreapta. Aceste proceduri se construiesc în mod asemănător și nu mai sînt detaliate aici. Precizăm numai că procedurile *arc_down*, *arc_right* și *arc_left* analizează direcțiile posibile pentru prima deplasare elementară în sens trigonometric direct, după cum urmează:

- *arc_down* — direcțiile 4, 5, 6, 7, 0;
- *arc_right* — direcțiile 6, 7, 0, 1, 2;
- *arc_left* — direcțiile 2, 3, 4, 5, 6.

Analog, procedurile *arc_down_rev*, *arc_right_rev*, *arc_left_rev* analizează direcțiile posibile ale primei deplasări în sens trigonometric invers:

- *arc_down_rev* — direcțiile 0, 7, 6, 5, 4;
- *arc_right_rev* — direcțiile 2, 1, 0, 7, 6;
- *arc_left_rev* — direcțiile 6, 5, 4, 3, 2.

8.4.6. Trasarea curbei generale de gradul doi

Sîntem acum în măsură să abordăm rutina principală de trasare a unei curbe conice, reprezentată analitic printr–o ecuație de forma (4.1). Structura algoritmului general de trasare este destul de simplă, și cuprinde în principal următoarele trei faze:

1) La început se calculează cantitățile δ, Δ după formulele (4.2), și se separă trasarea conicelor degenerate ($\Delta = 0$). După cum se va arăta în secțiunea 8.4.7, în multe cazuri particulare de degenerare algoritmul principal de trasare funcționează corect; totuși, există situații cînd trasarea curbelor degenerate trebuie abordată separat.

Pentru algoritmul principal de trasare se asigură condiția $\Delta > 0$, eventual printr–o schimbare generală de semn a coeficienților $a_i, i = \overline{1,6}$.

2) În faza cea mai importantă a algoritmului, care începe cu Pasul 3, se calculează toate intersecțiile curbei (4.1) cu laturile ferestrei de vizualizare. Funcția $f(x,y)$ fiind de gradul doi în x și y, curba $f(x,y) = 0$ intersectează fiecare latură a ferestrei în 0, 1 sau 2 puncte; în total pot exista maximum 8 puncte de intersecție (fig. 8–26).

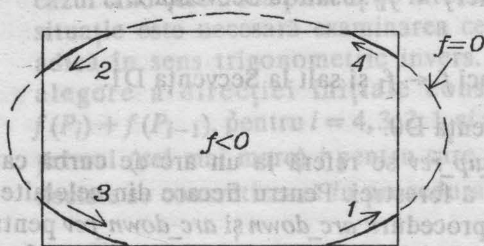


Figura 8–26.

Pot exista maximum 8 puncte de intersecţie ale conicei (4.1) cu marginile ferestrei dreptunghiulare de vizualizare, şi maximum 4 arce de curbă în interiorul ferestrei, care sînt trasate în ordinea indicată.

Pentru fiecare punct de intersecţie, se cercetează semnul funcţiei f la dreapta arcului de curbă care porneşte din acest punct, şi este orientat spre interiorul ferestrei. Dacă $f < 0$ la dreapta curbei, atunci punctul de intersecţie este neglijat, deoarece arcul de curbă trebuie trasat începînd de la celălalt capăt, care corespunde altui punct de intersecţie. Dacă $f > 0$ la dreapta curbei, atunci arcul de curbă care porneşte din punctul de intersecţie respectiv este trasat cu ajutorul uneia din procedurile dezvoltate în secţiunea 8.4.5. În cazul în care există două puncte de intersecţie, unul şi numai unul dintre acestea îndeplineşte condiţia de mai sus cu privire la semnul lui f.

Modul de calculare a punctelor de intersecţie ale curbei ideale cu marginile ferestrei de vizualizare necesită o analiză atentă, pentru a evita erorile de plasare care pot rezulta din structura discretă a mediului de afişare. Ne vom referi la intersecţia cu marginea de jos ($y = ymin$), pentru celelalte margini problema rezolvîndu-se în mod analog.

Pentru a determina abscisele punctelor de intersecţie ale curbei $f(x,y) = 0$ cu dreapta $y = ymin$, se calculează aproximări întregi prin lipsă pentru rădăcinile ecuaţiei în x:

$$f(x,y) + f(x,y-1) = 2a_1x^2 + a_2x(2y-1) + a_3(2y^2-2y+1) + \\ + 2a_4x + a_5(2y-1) + 2a_6 = 0 \quad (4.21)$$

unde $y = ymin$. Această condiţie conduce la rezultatul corect, spre deosebire de ecuaţia teoretică $f(x,ymin) = 0$, în spaţiul discret de afişare în care interesează numai coordonatele întregi ale punctelor generate. De exemplu, în cazul ilustrat în fig. 8–27 în care curba intersectează orizontala sub un unghi foarte ascuţit, ecuaţia (4.21) conduce la adresa x a primului punct generat, care nu coincide cu abscisa punctului teoretic de intersecţie.

Ecuaţia (4.21) este o consecinţă a faptului că $f(x,ymin) + f(x,ymin-1) < 0$ (respectiv > 0) pentru toate punctele $(x,ymin)$ mai apropiate (respectiv mai depărtate) de curbă decît $(x,ymin-1)$. În cazul limită în care $f(x,ymin) + f(x,ymin-1) = 0$, avem $f(x,ymin) = -f(x,ymin-1)$, ceea ce înseamnă că punctele $(x,ymin)$ şi $(x,ymin-1)$ sînt (aproximativ) egal

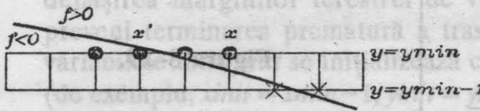


Figura 8–27.

Ecuaţia (4.21) furnizează abscisa x a primului punct generat, care nu coincide întotdeauna cu abscisa x' a punctului teoretic de intersecţie (în aproximaţie întreagă).

depărtate de curba ideală, în conformitate cu ipoteza I_1 discutată în secţiunea 8.4.2. Prin urmare ecuaţia (4.21) furnizează direct adresa x începînd de la care punctele care trebuie generate intră în fereastră (eventual cu o eroare de un punct spre stînga, datorită aproximării prin lipsă).

Dacă curba $f(x,y) = 0$ intersectează orizontala $y = ymin$ sub un unghi apropiat de 90°, atunci ecuaţia (4.21) conduce la acelaşi rezultat ca ecuaţia $f(x,ymin) = 0$ (în aproximaţie întreagă).

Condiţia (4.21) este în particular foarte adecvată pentru cazurile în care curba $f(x,y) = 0$ este aproape tangentă la dreapta $y = ymin$. În aceste cazuri, ecuaţia (4.21) nu are soluţii dacă toate punctele care trebuie generate sînt în interiorul ferestrei de vizualizare. De exemplu, parabola $f(x,y) = (x-10)^2-12y-4 = 0$ intersectează axa $y = 0$ în punctele $(8,0)$ şi $(12,0)$; pe de altă parte, ecuaţia $f(x,0) + f(x,-1) = 2(x-10)^2+4 = 0$ nu are soluţii, în conformitate cu faptul că nici unul din punctele din rastru care trebuie generate nu coboară sub axa $y = 0$ (fig. 8–28). Acest fapt va avea ca rezultat trasarea întregii curbe printr–un singur arc, pornind de la unul din punctele de intersecţie cu celelalte margini $y = ymax$, $x = xmin$ sau $x = xmax$.

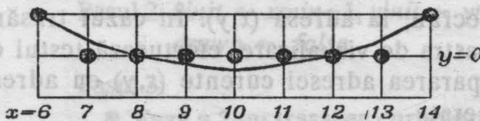


Figura 8–28.

Parabola $(x-10)^2-12y-4 = 0$ intersectează axa $y = 0$, dar toate punctele generate au ordonate $y \geq 0$.

3) În faza a 3–a, care începe cu Pasul 17 al algoritmului, se trasează curbele care nu intersectează marginile ferestrei de vizualizare: cercuri sau elipse complet incluse în fereastră. Această situaţie se detectează prin condiţia *drawn* = *.false.* Variabila logică globală *drawn*, iniţializată cu valoarea *.false.* în faza 1, ia valoarea *.true.* dacă cel puţin un arc de curbă este efectiv generat în faza 2.

Cercurile şi elipsele complet incluse în fereastră se trasează în modul

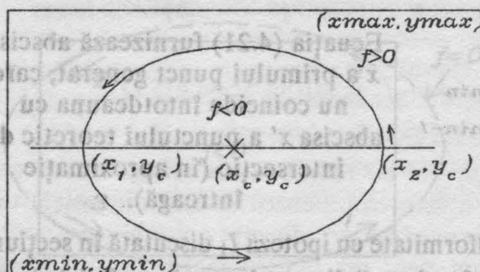


Figura 8–29.

Trasarea curbelor complet incluse în fereastra de vizualizare.

următor (fig. 8–29): se calculează ordonata y_c a centrului curbei, după formula (4.23), apoi se calculează abscisele x_1, x_2, $x_1 \leq x_2$ ale punctelor de intersecţie ale curbei cu orizontala $y = y_c$, iar în final se apelează procedura *arc_up* pentru trasarea efectivă, începînd cu punctul de intersecţie din dreapta (x_2, y_c).

În acest caz, procesul iterativ de generare a punctelor se termină după parcurgerea completă a curbei închise, deci atunci cînd se ajunge din nou la primul punct generat. Înainte de începerea trasării, adresa primului punct generat se memorează cu ajutorul variabilelor *xinit, yinit*. Condiţia de terminare a trasării este testată în rutina *plot1*, care este apelată pentru generarea efectivă a fiecărui punct, şi care este descrisă în continuare. Variabila logică *second*, iniţializată la valoarea *.false.* la intrarea în faza a 3–a a algoritmului, este utilizată în rutina *plot1* pentru oprirea trasării atunci cînd se ajunge a doua oară la adresa (*xinit, yinit*).

Rutina *plot1* (*x*, *y*)

Generează un punct pe ecran, la adresa (*x*,*y*). În cazul trasării curbelor închise incluse în fereastra de vizualizare, efectuează testul de terminare a trasării prin compararea adresei curente (*x*,*y*) cu adresa (*xinit, yinit*) a primului punct generat.

Pasul 1: Dacă $x \neq xinit$, salt la Pasul 2;
dacă $y \neq yinit$, salt la Pasul 2;
dacă *second* = *.true.*, atunci stop;
second ← *.true.*
Pasul 2: *call plot* (*x*, *y*);
return.

Rutina *plot*1 este utilizată și în faza a 2–a a algoritmului, cînd terminarea trasării unui arc de curbă se detectează într–un mod diferit, și anume prin depășirea marginilor ferestrei de vizualizare (secțiunea 8.4.4). Pentru a preveni terminarea prematură a trasării datorită testului din rutina *plot*1, variabilele *xinit*, *yinit* se inițializează cu adresa unui punct imposibil de atins (de exemplu, $xinit = xmin-1$, $yinit = ymin-1$) la intrarea în faza a 2–a.

După aceste pregătiri, sîntem în măsură să construim modulul principal al algoritmului de trasare a curbelor conice plane, reprezentate printr–o ecuație de gradul doi de forma generală (4.1). Alte precizări sînt adăugate în cuprinsul algoritmului și marcate cu semnul „•".

Algoritmul D

Trasarea incrementală a unei curbe plane de gradul doi, de ecuație $f(x,y) = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6 = 0$. Date de intrare: coeficienții $a_i, i = \overline{1,6}$ și marginile ferestrei de vizualizare *xmin*, *ymin*, *xmax*, *ymax* (toate sub formă de numere întregi).

Pasul 1:

- **Faza 1: inițializări**
  $\delta \leftarrow a_2^2 - 4a_1a_3$;
  $\Delta \leftarrow a_1a_5^2 + a_3a_4^2 - a_2a_4a_5 + a_6\delta$;
  dacă $\Delta = 0$, trasează conică degenerată
- a se vedea secțiunea 8.4.7;
  dacă $\Delta > 0$, salt la Pasul 2;
  $a_i \leftarrow -a_i$ pentru $i = \overline{1,6}$;
  $\Delta \leftarrow -\Delta$
- a fost asigurată condiția $\Delta > 0$.

Pasul 2: $xinit \leftarrow xmin-1$, $yinit \leftarrow ymin-1$;
  $drawn \leftarrow$ *.false.*

Pasul 3:

- **Faza a 2–a:** trasarea curbelor care intersectează marginile ferestrei de vizualizare. Sînt trasate pe rînd toate arcele de curbă neîntrerupte care sînt incluse în fereastră.
  Dacă $a_1 \neq 0$, salt la Pasul 6.

Pasul 4:

- Curba intersectează cel mult într–un punct marginile de jos și de sus ale ferestrei
  $y \leftarrow ymin$;
  dacă $a_2(2y-1)+2a_4 = 0$, salt la Pasul 5

- În acest caz, $f(x, ymin) + f(x, ymin-1)$ nu se anulează, deci curba nu traversează orizontala $y = ymin$;

  $$x \leftarrow -\frac{a_3(2y^2-2y+1)+a_5(2y-1)+2a_6}{a_2(2y-1)+2a_4}$$

- Abscisa punctului de intersecţie, calculată din ecuaţia $f(x, ymin) + f(x, ymin-1) = 0$. Toate intersecţiile se calculează prin aproximare la numere întregi prin lipsă, dar acest aspect nu este exprimat explicit;

  dacă $a_1(2x+1)+a_2y+a_4 < 0$, salt la Pasul 5

- trasarea efectivă are loc numai dacă $f > 0$ la dreapta direcţiei de înaintare, ceea ce în acest caz este echivalent cu $f(x+1, y) > f(x, y)$;

  call arc_up.

Pasul 5: $y \leftarrow ymax$;

  dacă $a_2(2y+1)+2a_4 = 0$, salt la Pasul 10

- $f(x, ymax) + f(x, ymax+1) \neq 0$, deci curba nu traversează orizontala $y = ymax$;

  $$x \leftarrow -\frac{a_3(2y^2+2y+1)+a_5(2y+1)+2a_6}{a_2(2y+1)+2a_4}$$

- abscisa punctului de intersecţie rezultă din ecuaţia $f(x, ymax) + f(x, ymax+1) = 0$;

  dacă $a_1(2x+1)+a_2y+a_4 > 0$, salt la Pasul 10

- $f > 0$ la dreapta direcţiei de înaintare $\Leftrightarrow f(x+1, y) < f(x, y)$;

  call arc_down;

  salt la Pasul 10.

Pasul 6:

- Curba intersectează marginile de jos şi de sus în cel mult două puncte;

  $y \leftarrow ymin$;

  $D \leftarrow [a_2(2y-1)+2a_4]^2 - 8a_1[a_3(2y^2-2y+1)+a_5(2y-1)+2a_6]$;

  dacă $D < 0$, salt la Pasul 8

- ecuaţia $f(x, ymin) + f(x, ymin-1) = 0$ nu are rădăcini reale;

  $$x_1 \leftarrow \frac{-[a_2(2y-1)+2a_4] - \sqrt{D}}{4a_1}, x_2 \leftarrow \frac{-[a_2(2y-1)+2a_4] + \sqrt{D}}{4a_1};$$

  dacă $x_1 > x_2$, atunci $t \leftarrow x_1, x_1 \leftarrow x_2, x_2 \leftarrow t$

- au fost calculate rădăcinile $x_1, x_2, x_1 \leq x_2$ ale ecuaţiei $f(x, ymin) + f(x, ymin-1) = 0$, care sînt abscisele punctelor de intersecţie cu orizontala $y = ymin$;

  dacă $a_1(2x_2+1)+a_2y+a_4 < 0$, salt la Pasul 7

- pentru arcul de curbă care porneşte din (x_2, y), $f > 0$ la dreapta

directiei de înaintare $\Leftrightarrow f(x_2+1,y) > f(x_2,y)$;
$x \leftarrow x_2$;
call arc_up

- conform situației din fig. 8–24;
salt la Pasul 8.

Pasul 7: $x \leftarrow x_1$;
call arc_up_rev

- conform situației din fig. 8–25.

Pasul 8: $y \leftarrow ymax$;
$D \leftarrow [a_2(2y+1)+2a_4]^2 - 8a_1[a_3(2y^2+2y+1)+a_5(2y+1)+2a_6]$;
dacă $D < 0$, salt la Pasul 10

- ecuația $f(x,ymax) + f(x,ymax+1) = 0$ nu are rădăcini reale;

$$x_1 \leftarrow \frac{-[a_2(2y+1)+2a_4] - \sqrt{D}}{4a_1}, x_2 \leftarrow \frac{-[a_2(2y+1)+2a_4] + \sqrt{D}}{4a_1};$$

dacă $x_1 > x_2$, atunci $t \leftarrow x_1, x_1 \leftarrow x_2, x_2 \leftarrow t$

- au fost calculate rădăcinile $x_1,x_2,x_1 \le x_2$ ale ecuației $f(x,ymax) + f(x,ymax+1) = 0$, care sînt abscisele punctelor de intersecție cu orizontala $y = ymax$;
dacă $a_1(2x_2+1)+a_2y+a_4 < 0$, salt la Pasul 9

- pentru arcul de curbă care pornește din (x_2,y), $f > 0$ la dreapta direcției de înaintare $\Leftrightarrow f(x_2+1,y) < f(x_2,y)$;
$x \leftarrow x_1$;
call arc_down;
salt la Pasul 10.

Pasul 9: $x \leftarrow x_2$;
call arc_down_rev.

Pasul 10:

- Se calculează acum intersecțiile cu marginile verticale ale ferestrei de vizualizare.
Dacă $a_3 \neq 0$, salt la Pasul 13.

Pasul 11:

- Curba intersectează cel mult într–un punct marginile din stînga și din dreapta;
$x \leftarrow xmin$
dacă $a_2(2x-1)+2a_5 = 0$, salt la Pasul 12;

$$y \leftarrow -\frac{a_1(2x^2-2x+1)+a_4(2x-1)+2a_6}{a_2(2x-1)+2a_5}$$

- ordonata punctului de intersecție, calculată din ecuația

$f(xmin, y) + f(xmin-1, y) = 0$;
dacă $a_3(2y+1)+a_2x+a_5 > 0$, salt la Pasul 12

- trasarea efectivă are loc numai dacă $f > 0$ la dreapta direcției de înaintare, ceea ce în acest caz este echivalent cu $f(x,y+1) < f(x,y)$;
  call arc_right.

Pasul 12: $x \leftarrow xmax$;
dacă $a_2(2x+1)+2a_5 = 0$, stop;
$$y \leftarrow -\frac{a_1(2x^2+2x+1)+a_4(2x+1)+2a_6}{a_2(2x+1)+2a_5};$$
dacă $a_3(2y+1)+a_2x+a_5 < 0$, stop;
call arc_left;
stop

- în acest caz algoritmul nu mai intră în faza a 3-a, în care sînt trasate numai curbe de tip eliptic, deoarece $a_1 \neq 0$, $a_3 \neq 0$ pentru acest gen de curbe.

Pasul 13:

- Curba intersectează marginile din stînga și din dreapta în cel mult două puncte
  $x \leftarrow xmin$;
  $D \leftarrow [a_2(2x-1)+2a_5]^2 - 8a_3[a_1(2x^2-2x+1)+a_4(2x-1)+2a_6]$;
  dacă $D < 0$, salt la Pasul 15;
  $$y_1 \leftarrow \frac{-[a_2(2x-1)+2a_5] - \sqrt{D}}{4a_3},\ y_2 \leftarrow \frac{-[a_2(2x-1)+2a_5] + \sqrt{D}}{4a_3};$$
  dacă $y_1 > y_2$, atunci $t \leftarrow y_1, y_1 \leftarrow y_2, y_2 \leftarrow t$;
  dacă $a_3(2y_2+1)+a_2x+a_5 < 0$, salt la Pasul 14;
  $y \leftarrow y_1$;
  call arc_right;
  salt la Pasul 15.

Pasul 14: $y \leftarrow y_2$;
call arc_right_rev.

Pasul 15: $x \leftarrow xmax$;
$D \leftarrow [a_2(2x+1)+2a_5]^2 - 8a_3[a_1(2x^2+2x+1)+a_4(2x+1)+2a_6]$;
dacă $D < 0$, salt la Pasul 17;
$$y_1 \leftarrow \frac{-[a_2(2x+1)+2a_5] - \sqrt{D}}{4a_3},\ y_2 \leftarrow \frac{-[a_2(2x+1)+2a_5] + \sqrt{D}}{4a_3};$$
dacă $y_1 > y_2$, atunci $t \leftarrow y_1, y_1 \leftarrow y_2, y_2 \leftarrow t$;
dacă $a_3(2y_2+1)+a_2x+a_5 < 0$, salt la Pasul 16;
$y \leftarrow y_2$;

call arc_left;
salt la Pasul 17.

Pasul 16: $y \leftarrow y_1$;
call arc_left_rev.

Pasul 17:

- **Faza a 3–a: trasarea curbelor închise de tip eliptic, care sînt complet incluse în fereastra de vizualizare**

Dacă *drawn* = *.true.*, stop

- cel puțin un arc de curbă a fost trasat în faza a 2–a;

dacă $\delta \geq 0$, stop

- curba este de tip hiperbolic sau parabolic, exterioară ferestrei de vizualizare;

$$y \leftarrow \frac{2a_1a_5 - a_2a_4}{\delta}$$

- ordonata centrului curbei de tip eliptic, care este și ordonata primului punct generat;

dacă $y < ymin$ sau $y > ymax$, stop

- curba este exterioară ferestrei;

$D \leftarrow (a_2y+a_4)^2 - 4a_1(a_3y^2+a_5y+a_6)$;
dacă $D < 0$, stop

- în acest caz ecuația nu reprezintă o curbă reală;

$$x_1 \leftarrow \frac{-(a_2y+a_4) - \sqrt{D}}{2a_1},\ x_2 \leftarrow \frac{-(a_2y+a_4) + \sqrt{D}}{2a_1};$$

dacă $x_1 > x_2$, atunci $t \leftarrow x_1, x_1 \leftarrow x_2, x_2 \leftarrow t$

- au fost calculate intersecțiile (x_1,y) și (x_2,y), $x_1 \leq x_2$ ale curbei cu orizontala care trece prin centru — fig. 8–29;

dacă $x_2 < xmin$ sau $x_2 > xmax$, stop

- curba este exterioară ferestrei, deși centrul curbei este în interiorul ei;

dacă $x_1 \neq x_2$, salt la Pasul 18;
call plot (x_2,y)

- în acest caz curba este aproximată printr–un singur punct;

stop.

Pasul 18: $x \leftarrow x_2$

- se pornește de la punctul de intersecție din dreapta — fig. 8–29;

$f \leftarrow a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6$;
$fx \leftarrow 2a_1x + a_2y + a_4$;
$f_0 \leftarrow f + fx + a_1$;
dacă $f+f_0 < 0$, atunci $x \leftarrow x + 1$

- corectare a poziției punctului inițial, asemănătoare cu cea

efectuată la începutul procedurii *arc_up*;
xinit ← *x*, *yinit* ← *y*

- se memorează adresa primului punct generat;
second ← *.false.*; salt la procedura *arc_up*
- pentru curbele închise, momentul terminării procesului de generare se detectează în rutina *plot1*, atunci cînd se ajunge a doua oară la adresa (*xinit*, *yinit*).

8.4.7. Curbe degenerate

După cum s-a dovedit în secțiunea 8.4.1, dacă $\Delta = 0$, ecuația (4.1) nu reprezintă o curbă propriu-zisă; ea se descompune în două ecuații de gradul întîi cu coeficienți în general complecși, care reprezintă două drepte plane în cazul cînd toți acești coeficienți sînt reali. Algoritmul de trasare dezvoltat în secțiunile precedente funcționează corect și poate fi aplicat pentru cazurile de degenerare cînd cele două drepte sînt paralele sau se intersectează în afara ferestrei de vizualizare. Dacă dreptele sînt concurente în interiorul ferestrei, algoritmul general funcționează aproape corect, în sensul că apar neliniarități de cîteva puncte în vecinătatea punctului de intersecție (fig. 8-30).

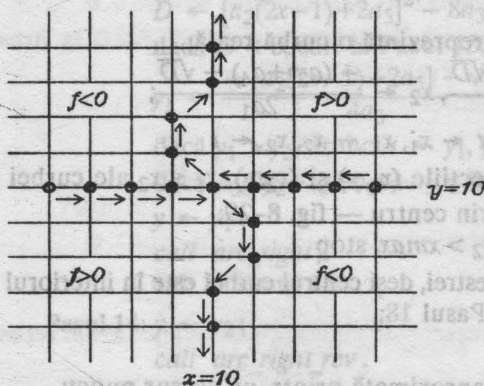


Figura 8–30. Rezultatul aplicării algoritmului general de trasare pentru curba degenerată $f(x,y) = xy - 10x - 10y + 100 = (x-10)(y-10) = 0$. Punctul (10, 10) este generat de două ori.

Algoritmul nu poate fi aplicat pentru trasarea curbelor degenerate în două drepte confundate (de exemplu $f(x,y) = x^2 - 2xy + y^2 = (x-y)^2 = 0$), deoarece nu poate fi asigurată condiția $f(x,y) < 0$ la stînga direcției de înaintare.

Pentru trasarea corectă a curbelor degenerate este posibilă și o altă metodă, care se impune în cazurile în care cele două drepte sînt confundate sau concurente în interiorul ecranului: se calculează coeficienții ecuațiilor celor două drepte, în funcție de coeficienții ecuației inițiale (4.1), și se aplică de două ori algoritmul general de trasare pentru cele două ecuații

obținute. Este evident că algoritmul poate fi aplicat pentru trasarea unei drepte ($a_1 = a_2 = a_3 = 0$), cu condiția ca ceilalți coeficienți a_4, a_5, a_6 să fie întregi sau raționali. După cum se va arăta în continuare, există cazuri particulare de degenerare cînd cele două drepte au coeficienți iraționali; în aceste cazuri este necesară înlocuirea prealabilă a coeficienților iraționali prin aproximări raționale suficient de precise.

În continuare prezentăm o analiză mai detaliată a cazurilor de degenerare a curbelor plane de gradul doi, cu deducerea expresiilor analitice ale celor două drepte componente; nu mai insistăm asupra detaliilor algoritmului de trasare pentru fiecare caz în parte, care nu ridică probleme de implementare deosebite.

Condiția necesară și suficientă de degenerare este (secțiunea 8.4.1):

$$\Delta = a_1a_5^2 + a_3a_4^2 - a_2a_4a_5 + (a_2^2 - 4a_1a_3)\, a_6 = 0\,. \tag{4.22}$$

Cazul 1. $\delta = a_2^2 - 4a_1a_3 \neq 0$. În acest caz conica are centru unic, de coordonate:

$$x_c = \frac{2a_3a_4 - a_2a_5}{a_2^2 - 4a_1a_3}, \quad y_c = \frac{2a_1a_5 - a_2a_4}{a_2^2 - 4a_1a_3} \tag{4.23}$$

și trece prin centru: se poate arăta că din (4.22) și (4.23) rezultă $f(x_c, y_c) = 0$ (centrul reprezintă de fapt punctul de intersecție a celor două drepte componente). Deoarece avem:

$$\begin{aligned} f(x+x_c, y+y_c) &= a_1x^2 + a_2xy + a_3y^2 + (2a_1x_c + a_2y_c + a_4)x + \\ &+ (a_2x_c + 2a_3y_c + a_5)y + f(x_c, y_c) = \\ &= a_1x^2 + a_2xy + a_3y^2, \end{aligned}$$

rezultă că ecuația curbei se poate scrie sub forma:

$$f(x,y) = a_1(x-x_c)^2 + a_2(x-x_c)(y-y_c) + a_3(y-y_c)^2 = 0\,. \tag{4.24}$$

1.1 • Dacă $\delta = a_2^2 - 4a_1a_3 < 0$, atunci ecuația (4.24) nu poate fi satisfăcută decît pentru $x = x_c$, $y = y_c$. Prin urmare, în acest caz conica se reduce la un singur punct (x_c, y_c), ale cărui coordonate sînt date de formulele (4.23).

1.2 • Dacă $\delta = a_2^2 - 4a_1a_3 > 0$, atunci conica se reduce la două drepte concurente, de ecuații:

$$2a_1x + (a_2 \pm \sqrt{\delta})y + \left(a_4 + \frac{a_2a_4 - 2a_1a_5}{\pm\sqrt{\delta}}\right) = 0 \tag{4.25}$$

$$(a_2\pm\sqrt{\delta})x+2a_3y+\left(a_5+\frac{a_2a_5-2a_3a_4}{\pm\sqrt{\delta}}\right) = 0$$

în care semnul din faţa lui $\sqrt{\delta}$ trebuie ales astfel încît să corespundă cu semnul efectiv al coeficientului a_2 (a se vedea secţiunea 8.4.1).

Cazul 2. $\delta = a_2^2 - 4a_1a_3 = 0$. Conica nu are centru unic. Rezultă în primul rînd că a_1 şi a_3 au acelaşi semn, deci putem presupune $a_1 \geq 0$, $a_3 \geq 0$ (eventual, este necesară o schimbare generală de semn a coeficienţilor, în faza iniţială).

Din $\Delta = 0$, $\delta = 0$ deducem succesiv următoarele relaţii între coeficienţi:

$$a_1a_5^2 + a_3a_4^2 - a_2a_4a_5 = 0 \Rightarrow a_1a_5^2 + a_3a_4^2 = a_2a_4a_5; \tag{4.26}$$

$$(a_2a_5 - 2a_3a_4)^2 = a_2^2a_5^2 + 4a_3^2a_4^2 - 4a_2a_3a_4a_5 =$$
$$= 4a_1a_3a_5^2 + 4a_3^2a_4^2 - 4a_3(a_1a_5^2 + a_3a_4^2) = 0 \Rightarrow$$
$$\Rightarrow a_2a_5 = 2a_3a_4; \tag{4.27}$$

$$(a_2a_4 - 2a_1a_5)^2 = a_2^2a_4^2 + 4a_1^2a_5^2 - 4a_1a_2a_4a_5 =$$
$$= 4a_1a_3a_4^2 + 4a_1^2a_5^2 - 4a_1(a_1a_5^2 + a_3a_4^2) = 0 \Rightarrow$$
$$\Rightarrow a_2a_4 = 2a_1a_5. \tag{4.28}$$

2.1 • Dacă $a_1 \neq 0$, adică $a_1 > 0$, ecuaţia $f(x,y) = 0$ se poate scrie sub forma echivalentă:

$$4a_1^2x^2 + 4a_1a_2xy + 4a_1a_3y^2 + 4a_1a_4x + 4a_1a_5y + 4a_1a_6 = 0 \Rightarrow$$
$$\Rightarrow 4a_1^2x^2 + 4a_1a_2xy + a_2^2y^2 + 4a_1a_4x + 2a_2a_4y + 4a_1a_6 = 0 \Rightarrow$$
$$\Rightarrow (2a_1x + a_2y)^2 + 2a_4(2a_1x + a_2y) + 4a_1a_6 = 0 \Rightarrow$$
$$\Rightarrow (2a_1x + a_2y + a_4)^2 = a_4^2 - 4a_1a_6 \tag{4.29}$$

în care am folosit relaţiile (4.28) şi $\delta = 0$. Distingem următoarele 3 subcazuri:

2.1.1 • Dacă, în plus, $a_4^2 > 4a_1a_6$, atunci conica se reduce la două drepte paralele, de ecuaţii:

$$2a_1x + a_2y + a_4 + \sqrt{a_4^2 - 4a_1a_6} = 0 \tag{4.30}$$
$$2a_1x + a_2y + a_4 - \sqrt{a_4^2 - 4a_1a_6} = 0\,.$$

În acest caz trasarea se poate efectua folosind algoritmul general dezvoltat în secțiunea 8.4.6, aplicat ecuației $f(x, y) = 0$ în forma inițială.

2.1.2 • Dacă $a_4^2 = 4a_1a_6$, conica se reduce la două drepte confundate de ecuație comună:

$$2a_1x + a_2y + a_4 = 0\,. \tag{4.31}$$

Trasarea curbei se poate efectua aplicînd algoritmul general pentru ecuația liniară (4.31), care are coeficienți întregi.

2.1.3 • Dacă $a_4^2 < 4a_1a_6$, ecuația $f(x, y) = 0$ nu reprezintă o curbă reală.

2.2 • Dacă $a_3 \neq 0$, adică $a_3 > 0$, ecuația $f(x, y) = 0$ se poate scrie sub forma echivalentă:

$$4a_1a_3x^2 + 4a_2a_3xy + 4a_3^2y^2 + 4a_3a_4x + 4a_3a_5y + 4a_3a_6 = 0 \Rightarrow$$

$$\Rightarrow\ a_2^2x^2 + 4a_2a_3xy + 4a_3^2y^2 + 2a_2a_5x + 4a_3a_5y + 4a_3a_6 = 0 \Rightarrow$$

$$\Rightarrow\ (a_2x + 2a_3y)^2 + 2a_5(a_2x + 2a_3y) + 4a_3a_6 = 0 \Rightarrow$$

$$\Rightarrow\ (a_2x + 2a_3y + a_5)^2 = a_5^2 - 4a_3a_6 \tag{4.32}$$

în care am folosit relațiile (4.27) și $\delta = 0$. Distingem din nou 3 subcazuri:

2.2.1 • Dacă, în plus, $a_5^2 > 4a_3a_6$, atunci conica se reduce la două drepte paralele, de ecuații:

$$a_2x + 2a_3y + a_5 + \sqrt{a_5^2 - 4a_3a_6} = 0 \tag{4.33}$$

$$a_2x + 2a_3y + a_5 - \sqrt{a_5^2 - 4a_3a_6} = 0\,.$$

Algoritmul general aplicat ecuației $f(x, y) = 0$ în forma inițială rezolvă corect problema trasării în acest caz.

2.2.2 • Dacă $a_5^2 = 4a_3a_6$, conica se reduce la două drepte confundate de ecuație comună:

$$a_2x + 2a_3y + a_5 = 0\,. \tag{4.34}$$

Trasarea curbei se poate efectua aplicînd algoritmul general pentru ecuația liniară (4.34), care are coeficienți întregi.

2.2.3 • Dacă $a_5^2 < 4a_3a_6$, ecuația $f(x, y) = 0$ nu reprezintă o curbă reală.

2.3 • Dacă $a_1 = a_3 = 0$, atunci din $\delta = 0$ rezultă $a_2 = 0$. Prin urmare, ecuația $f(x, y) = a_4x + a_5y + a_6 = 0$ este de gradul întîi și reprezintă o dreaptă, care poate fi trasată prin aplicarea algoritmului general.

8.4.8. Consideraţii de implementare

Algoritmul de trasare a curbelor plane de gradul doi, prezentat în secţiunile anterioare, a fost proiectat astfel încît să permită o implementare cît mai eficientă cu ajutorul microprocesoarelor actuale de uz general. Toate variabilele şi constantele folosite sînt întregi; în partea sa cea mai importantă din punct de vedere al eficienţei, care este bucla principală de generare iterativă a punctelor (secţiunea 8.4.4), algoritmul foloseşte numai operaţii de adunare, scădere şi comparaţie între numere întregi.

Mărimea clasei de curbe care pot fi generate este determinată de modul de alegere a reprezentărilor în memorie pentru variabilele şi constantele utilizate de algoritm. Pentru aplicaţiile practice curente, sînt suficiente şi sînt recomandate următoarele forme de reprezentare:

- coeficienţii a_1, a_2, a_3 se reprezintă pe 8 sau 16 biţi;
- coeficienţii a_4, a_5, se reprezintă pe 16 biţi;
- coeficientul a_6 se reprezintă pe 32 biţi;
- constanta δ se reprezintă pe 16 biţi;
- constanta Δ se reprezintă pe 32 biţi;
- variabilele de stare f, fx, fy şi variabilele temporare $t, f_0, f_1, \ldots f_7$ se reprezintă pe 16 biţi;
- variabila D, utilizată în calcularea punctelor de intersecţie cu marginile, se reprezintă pe 32 biţi.

În general, expresiile de gradul 1 sau 2 în coeficienţi şi argumente (de exemplu $\delta = a_2^2 - 4a_1a_3$, $fx = 2a_1x + a_2y + a_4$, $fy = a_2x + 2a_3y + a_5$, etc.) se calculează pe 16 biţi, iar expresiile de gradul 3 sau 4 (de exemplu Δ, D, etc.) se calculează pe 32 biţi. Variabila f, deşi corespunde unei expresii de gradul 3, se poate reprezenta pe 16 biţi, deoarece este întotdeauna egală cu valoarea funcţiei într-un punct din imediata vecinătate a curbei $f(x, y) = 0$.

Pentru mărirea eficienţei în execuţie, toate constantele derivate din coeficienţi, folosite în bucla principală de generare (secvenţele D0, D1, ... D7 din secţiunea 8.4.4), de exemplu $2a_1 + a_2$, $a_2 + 2a_3$, $a_1 + a_2 + a_3$, etc., trebuie calculate o singură dată (pe 16 biţi) la începutul programului. Este necesară alocarea în memorie a unui cuvînt de 16 biţi pentru fiecare din aceste constante.

Prin urmare, pentru implementarea eficientă a buclei principale de generare a punctelor, este suficientă aritmetica pe 16 biţi, în virgulă fixă, fără înmulţiri şi împărţiri.

La începutul trasării unui arc de curbă sînt necesare operaţii de înmulţire între numere întregi pentru iniţializarea variabilelor de stare (procedurile *arc_up*, *arc_up_rev*, etc.). În plus, calcularea punctelor de intersecţie ale curbei cu marginile ferestrei de vizualizare implică efectuarea

de înmulţiri, împărţiri şi extrageri de rădăcini pătrate, care trebuie să furnizeze partea întreagă a rezultatului exact. Dacă procesorul utilizat nu dispune de instrucţiuni de înmulţire şi împărţire, este necesară dezvoltarea unor rutine speciale pentru efectuarea acestor operaţii; de asemenea, este necesară o rutină specială pentru calcularea radicalilor. Eventual, este posibilă utilizarea subrutinelor corespunzătoare din biblioteca aritmetică a unui limbaj de nivel înalt. Eficienţa acestor operaţii aritmetice nu este importantă pentru performanţele globale ale algoritmului, deoarece ele apar numai în afara buclei principale de generare incrementală a punctelor.

8.5. Concluzii

Concluzia generală este că trasarea incrementală a curbelor de gradul întîi şi doi, pe dispozitive de afişare grafică cu rastru dreptunghiular, nu ridică probleme de implementare deosebite în cazul programării atente a rezultatelor teoretice. Pentru trasarea dreptelor şi cercurilor, este suficientă aritmetica pe 8 sau 16 biţi, în virgulă fixă, deoarece algoritmii respectivi utilizează numai operaţii de adunare, scădere şi comparaţie de numere întregi.

Pentru trasarea unei curbe generale de gradul doi, aritmetica în virgulă mobilă este bineînţeles de mare utilitate, dar nu este neapărat necesară. Implementări foarte eficiente sînt posibile cu ajutorul microprocesoarelor actuale, cu aritmetică în virgulă fixă, cu sau fără instrucţiuni de înmulţire şi împărţire.

8.6. Bibliografie

[1] **J.D. Foley, A. Van Dam**
Fundamentals of Interactive Computer Graphics, Addison Wesley, 1983

[2] **W.M. Newman, R.F. Sproull**
Principles of Interactive Computer Graphics, Second Edition, McGraw–Hill, 1979

[3] **J.E. Bresenham, R.A. Earnshaw, A.R. Forrest, R.J. Lansdown, M.L.V. Pitteway**
Theoretical Foundations of Computer Graphics and CAD, NATO ASI Series, Springer Verlag, 1988

[4] **J.E. Bresenham**
A Linear Algorithm for Incremental Digital Display of Circular Arcs, CACM 20, february 1977

[5] **Y. Suenaga, T. Kamae, T. Kabayashi**
A High–Speed Algorithm for the Generation of Straight Lines and Circular Arcs, IEEE Transactions on Computers, no. 10, october 1979

[6] * * * *Mică Enciclopedie Matematică,* Editura Tehnică, 1980

de înmulţiri, împărţiri şi extrageri de rădăcini pătrate, care trebuie să furnizeze partea întreagă a rezultatului exact. Dacă procesorul utilizat nu dispune de instrucţiuni de înmulţire şi împărţire, este necesară dezvoltarea unor rutine speciale pentru efectuarea acestor operaţii; de asemenea, este necesară o rutină specială pentru calcularea radicalilor. Eventual, este posibilă utilizarea subrutinelor corespunzătoare din biblioteca aritmetică a unui limbaj de nivel înalt. Eficienţa acestor operaţii aritmetice nu este importantă pentru performanţele globale ale algoritmului, deoarece ele apar numai în afara buclei principale de generare incrementală a punctelor.

8.5. Concluzii

Concluzia generală ce se desprinde este că trasarea incrementală a curbelor de gradul unu şi doi, pe dispozitive de afişare grafică cu rastru dreptunghiular, nu ridică probleme de implementare deosebite în cazul programării atente a rezultatelor teoretice. Pentru trasarea dreptelor şi cercurilor, este suficientă aritmetica pe 8 sau 16 biţi, în virgulă fixă, deoarece algoritmii respectivi utilizează numai operaţii de adunare, scădere şi comparaţie de numere întregi.

Pentru trasarea curbelor generale de gradul doi, aritmetica în virgulă mobilă este utilizabilă, aşa cum s-a arătat, dar nu este neapărat necesară, implementarea fiind foarte eficientă şi cu ajutorul microprocesoarelor actuale, cu aritmetică în virgulă fixă, cu sau fără instrucţiuni de înmulţire şi împărţire.

8.6. Bibliografie

[1] J.D. Foley, A. Van Dam — *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, 1983

[2] W.M. Newman, R.F. Sproull — *Principles of Interactive Computer Graphics*, Second Edition, McGraw-Hill, 1979

[3] J.E. Bresenham, R.A. Earnshaw, A.R. Forrest, K.J. Lindahl, *Fundamental Algorithms for Computer Graphics*, Springer Verlag, 1985

[4] J.E. Bresenham — *A Linear Algorithm for Incremental Digital Display of Circular Arcs*, CACM 20, February 1977

[5] Y. Suenaga, T. Kamae, T. Kobayashi — *A High-Speed Algorithm for the Generation of Straight Lines and Circular Arcs*, IEEE Transactions on Computers, no. 10, october 1979

[6] * * * — *Mică Enciclopedie Matematică*, Editura Tehnică, 1980

ÎN CURS DE APARIŢIE

Tudor Bălănescu **Horia Georgescu** **Liviu Sofonea** **Şerban Gavrilă** **Marian Gheorghe** **Ion Văduva**

PROGRAMAREA ÎN LIMBAJELE PASCAL ŞI TURBO PASCAL

Volumul I: Programarea în limbajul Pascal. Concepte fundamentale.
Volumul II: Programarea în limbajul TURBO PASCAL

Volumul I: Lucrarea este un ghid de învăţare a limbajului Pascal, îmbinînd într-un mod foarte accesibil teoria şi exerciţiile practice. Scopul lucrării este punerea în evidenţă a simplităţii şi clarităţii acestui limbaj foarte răspîndit, ca şi posibilitatea de a atinge o anumită disciplină a programării.

Lucrarea cuprinde:

- o introducere în care se prezintă cîteva exemple de programare în limbaj Pascal pentru a forma cititorului o imagine de ansamblu asupra acestuia;
- capitolul 1 în care se prezintă concepte de lexic şi sintaxă;
- capitolul 2 care abordează tipurile de date, variabile, constante şi expresii ale limbajului;
- capitolul 3 în care se prezintă instrucţiunile (simple şi structurate);
- capitolul 4 care se ocupă de noţiunile de procedură, funcţii program şi modul, ca mecanisme de abstractizare a algoritmilor;
- capitolul 5 prezintă metodologia lui Hoare de exprimare a raţionamentelor asupra corectitudinii programelor;
- capitolul 6 prezintă facilităţile de depanare simbolică ale compilatorului Pascal - Oregon.

Volumul II: Lucrarea prezintă mediul de dezvoltare a programelor Turbo-Pascal, versiunea 5.5 creat de firma Borland International.

În prima parte se face o introducere gradată, bazată pe numeroase ilustraţii, în principiile de funcţionare şi utilizare a mediului Turbo Pascal.

În următoarele capitole (1-4) sînt prezentate particularităţile limbajului de programare Turbo-Pascal, prin raportarea la descrierea conceptelor standard, conţinută în "Programarea în limbajul Pascal - Concepte fundamentale".

Cap. 5 tratează principiile de organizare a programelor precum şi tehnici de programare orientată pe obiecte.

Cap. 6 conţine descrierea unitaţilor standard. Sînt prezentate numeroase aplicaţii de utilizare a unitaţilor Graph şi Crt (editor grafic, scrierea textelor pe direcţii arbitrare, grafica Turtle).

Lucrarea mai conţine un număr de anexe cuprinzînd elemente ale limbajului şi comenzi de utilizare a mediului Turbo-Pascal.



Lucrarea se adresează studenţilor, absolvenţilor facultăţii de matematică şi ai secţiilor de specialitate din institutele tehnice şi economice.

- Obiectivul cărţii este acela de a reuni aspectele de implementare şi de aplicaţii pentru grafica pe calculator în limbajele moderne de programare **PASCAL** şi **C**, unele dintre subiectele abordate fiind în premieră la noi în ţară.
- Cartea se adresează studenţilor, chiar elevilor, cercetătorilor, cadrelor didactice şi tuturor utilizatorilor calculatoarelor electronice din domeniul cercetării şi proiectării asistate de calculator.
- Utilitatea cărţii rezultă din faptul că subiectele tratate sînt rezultatul colaborării dintre specialişti ce lucrează în învăţămînt, cercetare şi în domeniul elaborării de software, şi sînt susţinute de multe programe executate pe calculator, împreună cu imaginile grafice corespunzătoare.



- **VOLUMUL I**, intitulat **„IMPLEMENTARE"**, abordează particularităţile de implementare a aspectelor grafice atît pe **minicalculatoare** (compatibile **PDP**), cît şi pe **microcalculatoare** (compatibile **IBM-PC**).
- **VOLUMUL II**, intitulat **„APLICAŢII"**, abordează aplicaţii de grafică pe calculator în domeniile: teoria fractalilor, teoria curbelor şi suprafeţelor, geometria de tip „TURTLE", trasarea curbelor pe dispozitivele de afişare grafică.

Vol. II Lei 325, —

ISBN 973-31-0406-X
ISBN 973-31-0408-6